ISSUE 17 - OCT 2013

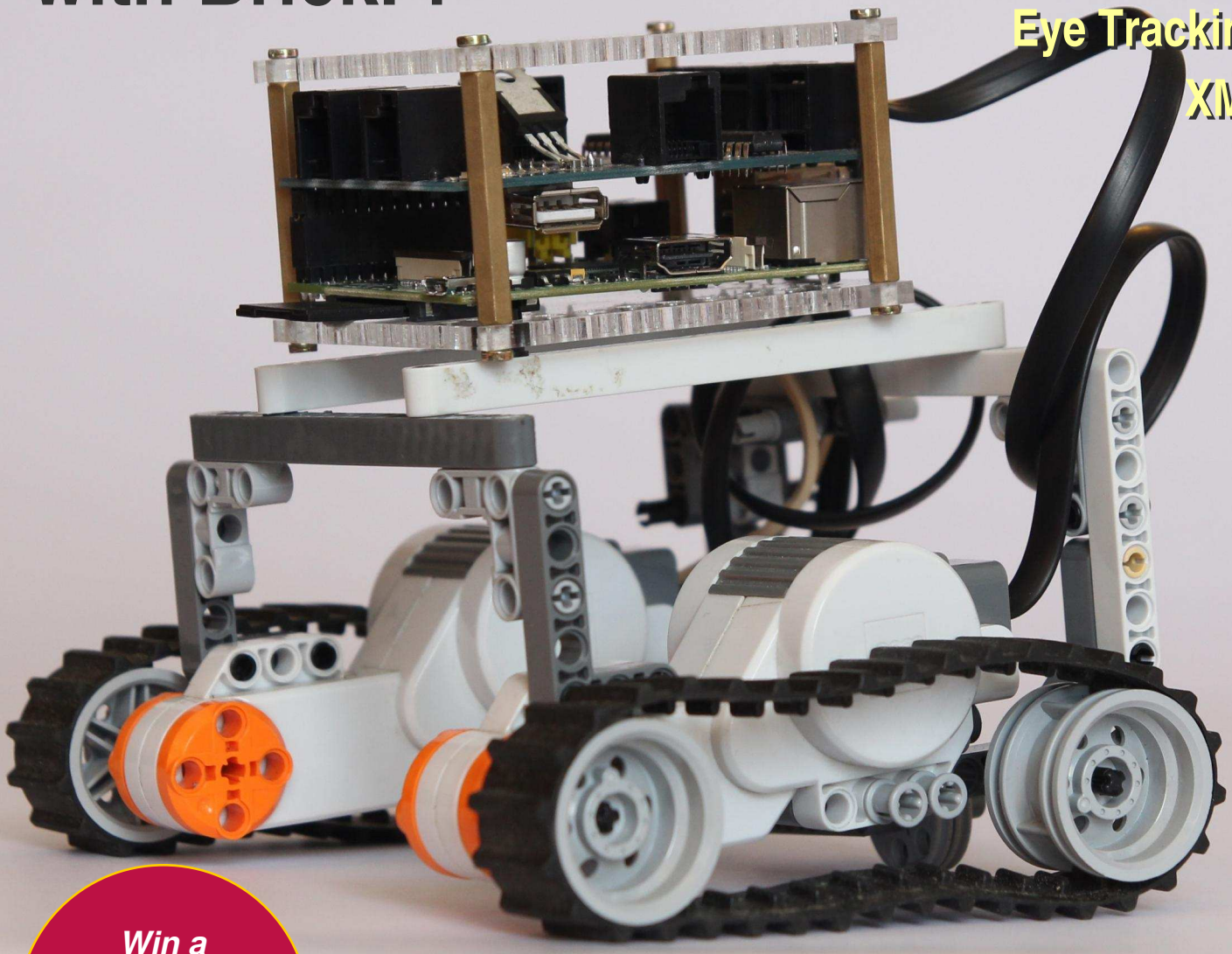
Get printed copies at themagpi.com

The MagPi™

A Magazine for Raspberry Pi Users

Robotics with BrickPi

I/O Expansion Boards
Radio Interface
C and Scratch
Eye Tracking
XML



Win a Raspberry Pi 16GB SD card breadboard & more

ISSN 2051-9990

Raspberry Pi is a trademark of The Raspberry Pi Foundation.
This magazine was created using a Raspberry Pi computer.

CC BY NC SA

The MagPi™

http://www.themagpi.com

Welcome to The MagPi - a free to download community driven magazine packed to the brim with all things Raspberry Pi.

As you can see from the cover, issue 17 has gone all robotic! We feature a great article from the team at Dexter Industries taking a look at their clever kit: the BrickPi. You may recognise the project from Kickstarter and the Foundation site. It is ideal for all levels of user and a great way to get your Raspberry Pi moving. Head over to page 4 for more information.

To add to the Sci-Fi theme we have included another article using the Raspberry Pi camera module. This month we introduce you to creating a program to track eye movements. We have more on NanPy and a great article in the series on the IO expander.

So far in The MagPi we have featured a multitude of articles which give reference to using WiFi to communicate between peripherals. Are you bored of being confined to your network? In this issue we set your Raspberry Pi free! Matthew looks at the ability for the Raspberry Pi to send and receive messages using radio waves. Team that with the BrickPi and the ability to track eyes and you could have yourself a very exciting project!

Due to its popular demand, we have more this month on XML and revive 'C Cave', describing unions and dynamic memory allocation. If this isn't enough, to top it all off we have a Scratch article teaching you how to create a program to fire ballistic missiles!

Of course, The MagPi recommends the use of goggles and a helmet when you try this out. *cough* *cough*

Hope you enjoy.

Ash Stone

Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration / Layout
W.H. Bell - Issue Editor / Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Tester / Proof Reading
Matt Judge - Website / Administration
Aaron Shaw - Layout
Colin Deady - Layout / Tester / Proof Reading
Shelton Caruthers - Tester / Proof Reading
Claire Price - Layout / Proof Reading
Matt Weaver - Layout
Bulent Tuncel - Layout
Matthew Watson - Layout
Phil Tesseyman - Tester
Courtney Blush - Proof Reading
Amy-Clare Martin - Proof Reading
Gerry Fillery - Proof Reading

Contents

4 **BRICKPI - LEGO® NXT INTERFACE**
Plug in LEGO® sensors and motors

8 **THE PICAMERAL EYETRACKER**
Tracking eye movement with the Raspberry Pi Camera Board

12 **USB ARDUINO LINK**
Part 3: Using a temperature sensor and Arduino analogue pins

16 **MUNTS I/O EXPANSION BOARD**
Part 2: Using the factory default firmware for the LPC1114 I/O processor exxpansion board

22 **LONG-RANGE WIRELESS**
Using pairs of radio modules with the Raspberry Pi

24 **PHYSICAL COMPUTING**
Buttons and switches with the Raspberry Pi

28 **<XML />**
Part 3: Controlling your Raspberry Pi from your smartphone using RasPiConnect

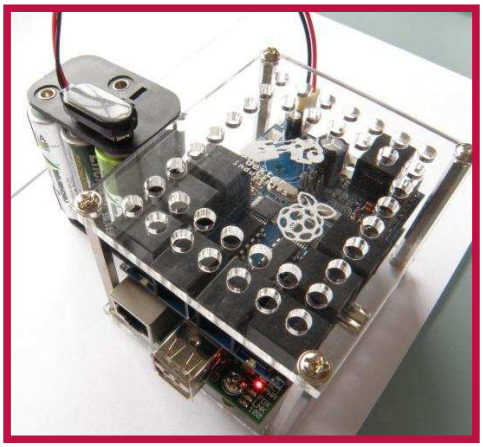
30 **COMPETITION**
Win a Raspberry Pi Model B, breadboard, 16GB NOOBS SD card and accessories

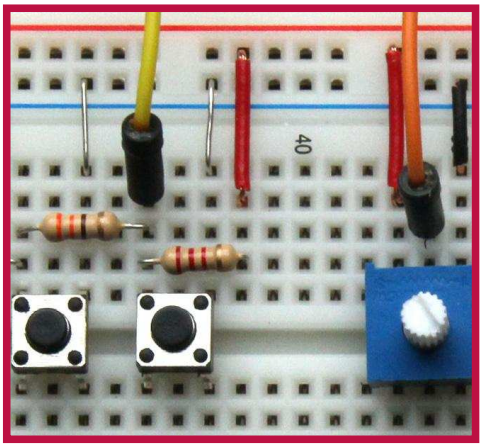
31 **THIS MONTH'S EVENTS GUIDE**
Pittsburgh USA, Exeter UK, Leicester UK, Stevenage UK, Miami USA

32 **THE C CAVE**
Memory management: unions and dynamic allocation

36 **SCRATCH PATCH**
Going ballistic: the physics of a cannon ball

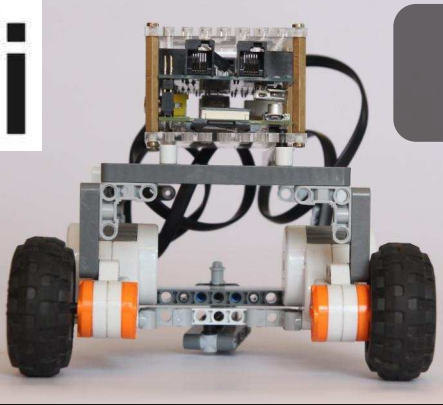
40 **FEEDBACK**
Have your say about The MagPi





Eye image in header on page 8 by Laitr Keiows (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons

BrickPi



LEGO NXT INTERFACE
Plug in LEGO® sensors and motors

Building bots with LEGO

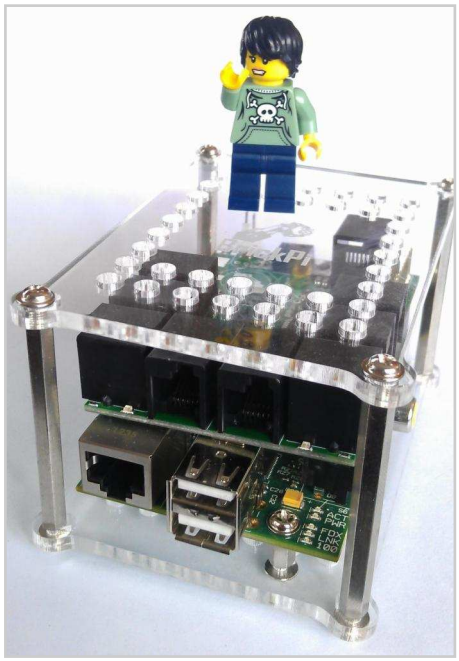
John Cole

Guest Writer

SKILL LEVEL : BEGINNER

What is the BrickPi?

The BrickPi is an interface between the Raspberry Pi and LEGO® Mindstorms sensors and motors, a portable power source for the Raspberry Pi, and a case that allows you to attach LEGO® motors, beams, sensors, and other parts.



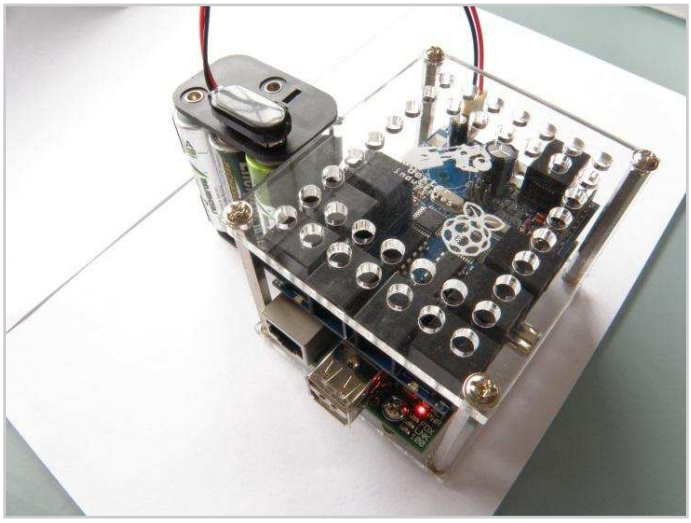
The BrickPi can be used to control up to four LEGO® NXT motors and read the motor encoder values. The motor encoder values measure how far the motor has turned. The BrickPi also has four sensor ports. These ports can read digital NXT sensors or analog sensors. There is also a fifth port that can be used with digital sensors. The BrickPi has a battery power unit that can provide power for the BrickPi, sensors, motors and Raspberry Pi. The power unit on the BrickPi can be used with a 9 to 12 Volt battery pack. The BrickPi system also includes an enclosure to which LEGO® Technic parts, sensors, and motors can be attached. The enclosure houses the Raspberry Pi and the BrickPi. It is made of acrylic and comes with evenly-spaced holes along the top and bottom that are big enough for Technic LEGO® pegs. The enclosure leaves the sides of the Raspberry Pi clear, allowing access to the USB ports.

Getting started

Follow the setup guide at
http://www.dexterindustries.com/BrickPi/getting-started/
to assemble the enclosure and for references to SD installation or software configuration. The BrickPi can be used with the BrickPi Raspbian image or by following the setup guide instructions to configure the standard Raspbian image. The BrickPi Raspbian image contains all of the configuration needed to get up and running as

quickly as possible.

The Raspberry Pi uses quite a lot of battery power. Therefore, it is a good idea to plug the Raspberry Pi into a mains adapter when it is being programmed and use 6 AA batteries for final remote robotics. The power consumption of the Raspberry Pi, BrickPi and associated motors or sensors varies according to the application. This means that the battery pack could last for hours or for 20 minutes, depending on what is connected. The battery life can be extended by using a Model A Raspberry Pi.



Working with the BrickPi

The BrickPi can be programmed using a keyboard, mouse and monitor plugged into the Raspberry Pi. However, sometimes a spare monitor is not available or the robot might be roaming around. Therefore, the BrickPi Raspbian image contains an automatically configured VNC server, SSH and a few other goodies. There is a short tutorial on connecting remotely to the Raspberry Pi using another computer at:
http://www.dexterindustries.com/BrickPi/getting-started/using-the-pi/

Programming

All the software for the BrickPi is open source, allowing it to be improved by the community. The physical design of the BrickPi board is also available. There are details of the firmware and software too. The hardware and the software are available on Github at:
https://github.com/DexterInd/BrickPi

There are drivers for C, Python, and MITs graphic language Scratch. The beauty of the hardware architecture and the Raspberry Pi is that any language that can run on the Raspberry Pi can be used to control a students robot.

Python interface - motor control

The python example below runs one motor on Port A forward for 3 seconds, and then backwards for 3 seconds. At its simplest, running a motor requires two lines of code: one to setup the motor, and one to set the motor power.

```
from BrickPi import *           # import BrickPi.py file to use BrickPi
operationsBrickPiSetup()        # setup the serial port for communication
BrickPi.MotorEnable[PORT_A] = 1 # Enable the Motor A
BrickPiSetupSensors()           # Send the properties of sensors to BrickPi
while True:                     # Keep looping until the program is killed
  print "Running Forward"
  BrickPi.MotorSpeed[PORT_A] = 200  # Set the speed of MotorA (-255 to 255)
  ot = time.time()
  while(time.time() - ot < 3):  # Run while loop for 3 seconds
    BrickPiUpdateValues()       # Ask BrickPi to update values for sensors/motors
    time.sleep(.1)              # sleep for 100 ms
  print "Running Reverse"
  BrickPi.MotorSpeed[PORT_A] = -200 # Set the speed of MotorA (-255 to 255)
  ot = time.time()
  while(time.time() - ot < 3):  # Run while loop for 3 seconds
    BrickPiUpdateValues()       # Ask BrickPi to update values for sensors/motors
    time.sleep(.1)              # sleep for 100 ms
```

Python interface - ultrasonic sensor

The LEGO® ultrasonic sensor can be attached to any of the four sensor ports and provides the distance an object is from the robot.

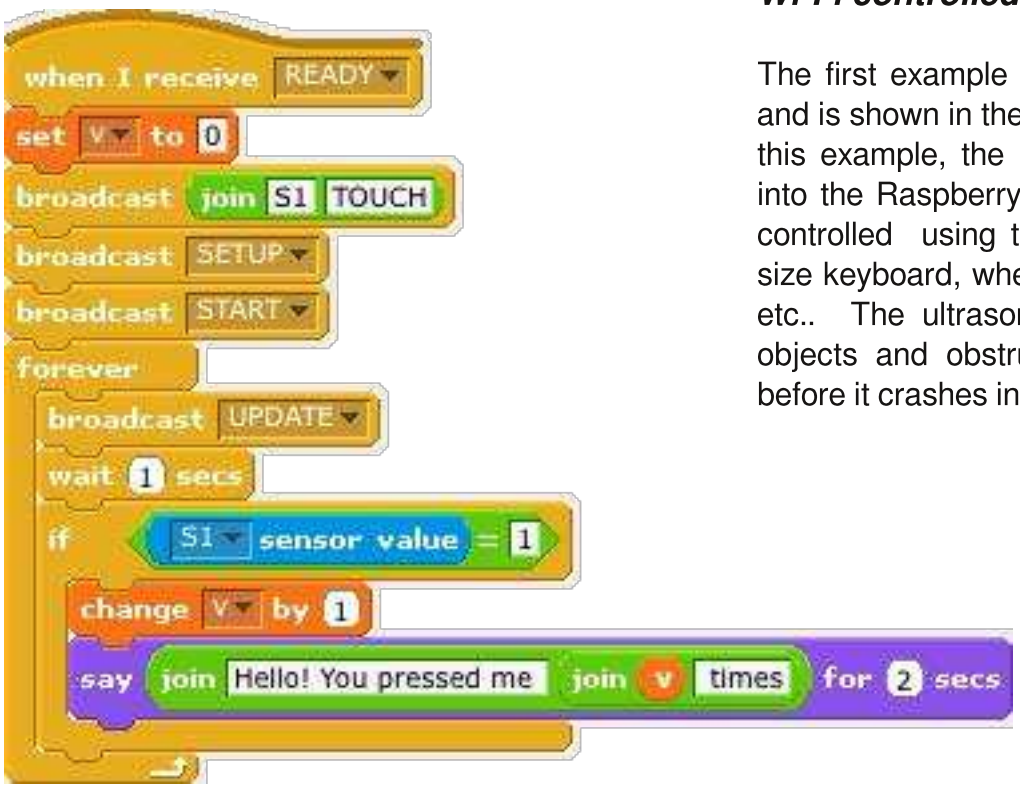
This is the common way to connect Scratch and Python together, relying on a piece of supporting Python to talk to the BrickPi. In this example, the sprite will count how many times the touch sensor has been touched.

```
from BrickPi import *           # import BrickPi.py file to use BrickPi operations
BrickPiSetup()                  # setup the serial port for communication
BrickPi.SensorType[PORT_1] = TYPE_SENSOR_ULTRASONIC_CONT # Set type of sensor at PORT_1
BrickPiSetupSensors()           # Send the properties of sensors to BrickPi
while True:                     # Keep looping until the program is killed
 result = BrickPiUpdateValues() # Ask BrickPi to update values for sensors/motors
 if not result:
   print BrickPi.Sensor[PORT_1] # BrickPi.Sensor[PORT] stores value obtained from sensor
 time.sleep(.01)                # sleep for 10 ms
```

The program above sets up the ultrasonic sensor on Port 1, and then reads the value from the sensor in a loop, printing the measured distance on the screen. Again, in its simplest form, running the ultrasonic sensor requires all of two lines of code: one initialises the sensor, and one refreshes the value the sensor is reading.

Scratch interface - Touch sensor

The Scratch interface is available through the network connection and broadcast messages.


when I receive READY
set v to 0
broadcast join S1 TOUCH
broadcast SETUP
broadcast START
forever
broadcast UPDATE
wait 1 secs
if S1 sensor value = 1
change v by 1
say join Hello! You pressed me join v times for 2 secs

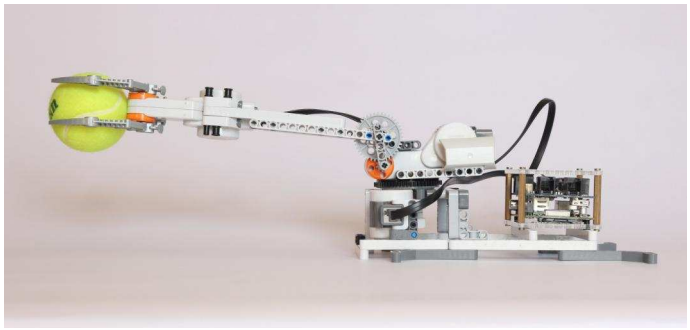
Cool projects

After testing out some of the examples using LEGO® sensors, there are some really creative project possibilities that use the horsepower of the Raspberry Pi. There are a few examples up already on the BrickPi Github and more will follow. So far, there are four example projects. These projects are provided with building instructions and Python code.

Wi-Fi controlled car

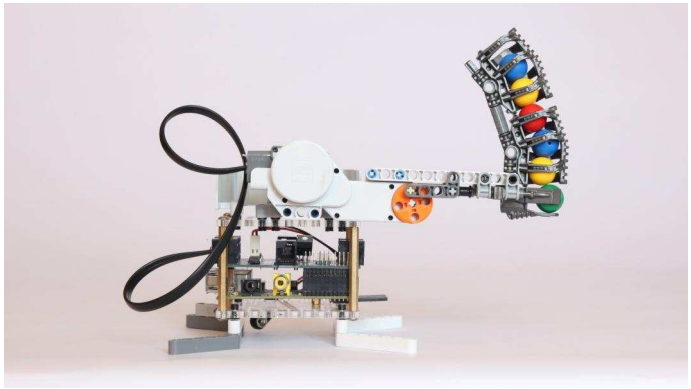
The first example is a car controlled with Wi-Fi and is shown in the header graphic on page 4. In this example, the robot is controlled by logging into the Raspberry Pi through SSH. The car is controlled using the numeric keypad on a full-size keyboard, where 8 is forward and 4 is left, etc.. The ultrasonic sensor is used to detect objects and obstructions, and to stop the car before it crashes into them.

Robotic arm



This example uses the BrickPi as a robotic arm for grabbing and moving materials. In this example, a keyboard connected to the Raspberry Pi is used to control the movement of the arm.

Shooter turret



This example shows how to build a turret ball shooter. The turret can rotate and is controlled by the mouse. Moving the mouse left and right changes the angle of the shooter, and the firing mechanism is controlled with a press of the mouse button. A great demonstration in class and fun in the office too.

Browser controlled tank

What better robot to build with tracks than a tank. A picture of the completed example model is on the cover of this Issue. The example sets up a webserver over Wi-Fi on the Raspberry Pi. The Raspberry Pi serves up a web page that has controls for moving the tank around and controlling the tank motion. The project could be extended with a webcam or other motors for interacting with the surroundings.

Buying motors and sensors

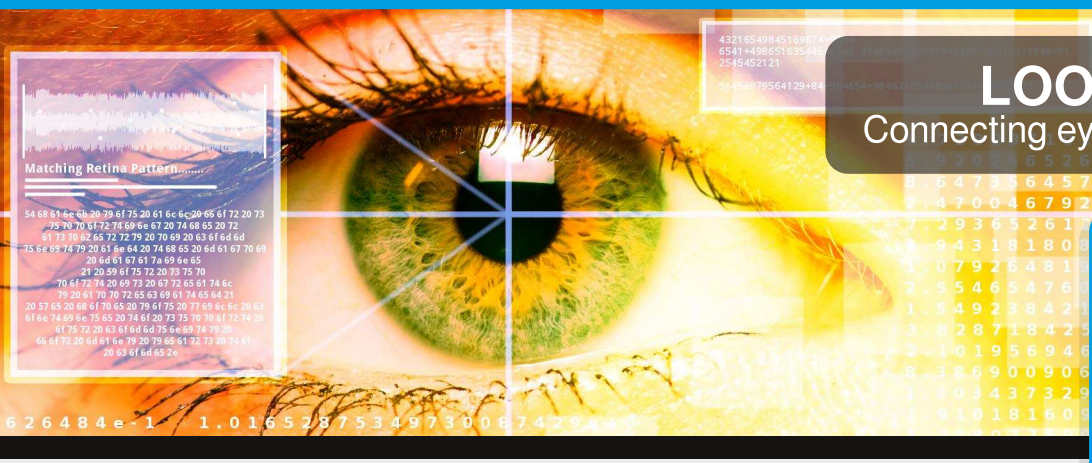
The BrickPi provides low cost access to robotics with LEGO® Mindstorms. It is not necessary to buy a full LEGO® Mindstorms kit. Instead, LEGO® sensors and motors can be bought from the LEGO® shop:
http://shop.lego.com/en-GB/Robotics-ByCategory
and used with any technical LEGO® set. The LEGO® shop and pick-a-brick web site do not sell the connecting wires in all countries. The connecting wires are available at:
http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=66
or distributors such as:
http://www.generationrobots.com/en/401377-cables-for-lego-mindstorms-nxt-robots-mindsensors.html
As well as the LEGO® sensors, other sensors and the BrickPi itself are available at:
http://www.dexterindustries.com/

Getting involved

The BrickPi is a great educational tool for young and old. The source code and hardware is open to allow the community to improve them. Try building your own project. There are BrickPi forums
http://www.dexterindustries.com/forum/?forum=brickpi
for communicating ideas and sharing projects with others. It would be great to see some more cool project ideas and hear how the BrickPi is being used.

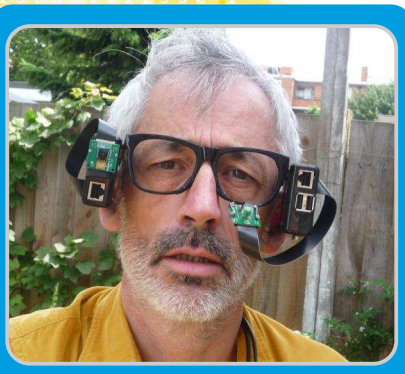
This is the first BrickPi article. Watch out for the next one, where the Scratch interface will be explained in detail.

LEGO® is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this site



The Picameral eye-tracker

SKILL LEVEL : ADVANCED



Jonathan Chetwynd

Guest Writer

The Picameral eye-tracker is hosted at http://www.gnote.org and is part of a larger project studying how we understand complex systems, my current visualisation is http://www.peepo.com.

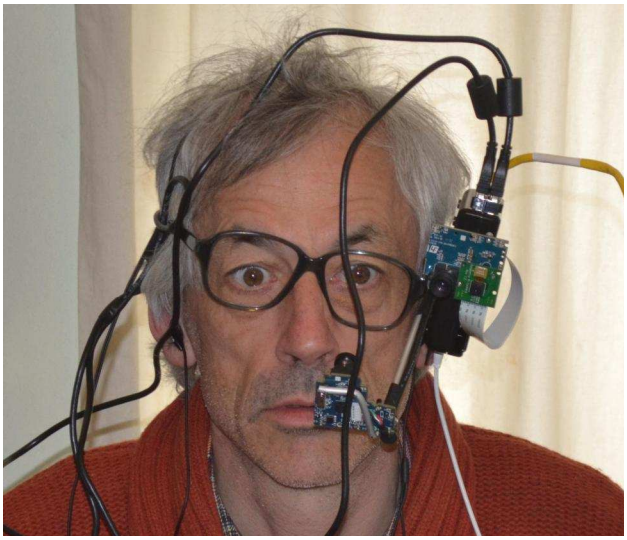
The Raspberry Pi computer enables, encourages and empowers the individual to explore and conduct independent research; so why make an eye tracker? The eye is a challenge as it is easily the fastest human organ, tracking its motion produces masses of data, and there are useful rules or heuristics to be derived from the various motions including saccades, glissades and fixations, or jerk, track and dwell.

Early eye-tracking research took place in a laboratory in a strictly controlled environment, perhaps using a forehead and chin rest to study reading skills or more recently web usability. It has only very recently become possible to make a cheap portable eyetracker. The total cost for this project should be ~£100 which includes 2x RaspberryPi with cameras, SD cards, cases, leads, etc. though you may already have some of this kit.

A portable eyetracker greatly broadens possible applications, perhaps recording and modelling social interactions at a party? whether side and rear-view mirrors are being observed regularly whilst driving? Is a soldier in the field using appropriate IED search techniques? You can easily imagine others.

There are some excellent resources available, the approach used builds from the excellent work of John Paulin Hansen and the team at ITU in Denmark who developed the Windows based GazeTracker; and the book Eye Tracking: by Holmqvist & Nyström 2011 provides a thorough introduction, without being overly academic.

Creating an eye-tracker is not itself a large problem, the difficulty is in considering the purposes for which you wish to use your device, and hence the peripheral devices and software required. For instance an 850ms dwell could be set to take a high quality still image using a second camera, here a cut down PS3 eyecam is connected with an RPi viewcam to a single RPi. The setup was tested and found to capture 200 images mostly of my choice, while cycling from Vauxhall to Kingston in about 45 minutes.



An earlier prototype had some load balancing problems that were improved in the final design.

Cameras have been improving fast, and the Raspberry Pi camera has many excellent qualities which we rely on. It can easily be adapted as a macro, which happens to focus at a useful distance with good quality video images. When mounted perpendicular to the camera board, the sensor is only a small obstruction and James Hughes implementation of Region Of Interest (ROI) enables digital cropping.

For more serious research, frame rates faster than 30 are desirable, and it is hoped that for formats considerably smaller than 1080HD, 60-90fps may be available shortly. The sensor is capable of 500fps for some formats, and this would allow one to estimate accelerations as well as velocities and locations.

As a project for one person, implementing an eyetracker requires a variety of skills, however the tasks might be split amongst a class of enthusiastic and bright teenagers!

Outline

The physical construction is relatively simple, and evident, in addition you will need: a database to store the data, a graphical tool for calibration, communications including a web server for ethernet and uart for serial port, perhaps audio feedback, and possibly a portable display.

Portable displays are ubiquitous, even HD is common, but HDMI inputs are rare to non-existent. If your project needs a portable interface the Kindle is cheap, and can use ethernet over usb to communicate.

Assuming you intend to visualise any results, which is not a requirement, I recommend html5 and NodeJS, a simple and powerful web server, with an up-to-date version of a web browser such as Mozilla Firefox, Chrome, Safari or Opera, or possibly IE11.

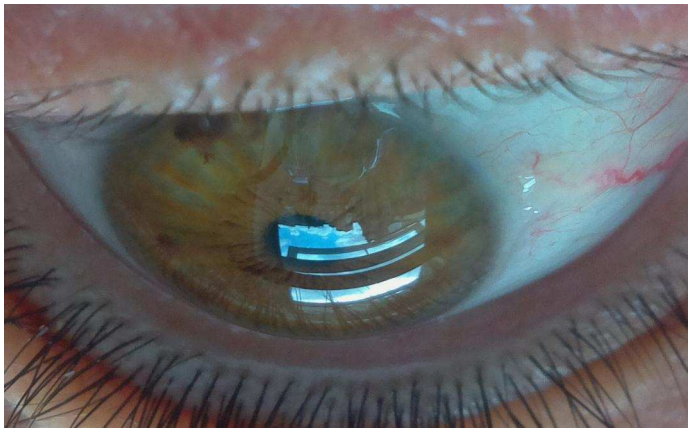
It may seem counter-intuitive, but whilst the RPi is capable of serving the eyetracking data, including video, it cannot currently run an html5 browser efficiently so you may need another box. I use a 5 year old mac-mini running Debian Wheezy for convenience.

During construction and development, communication relies on ssh over ethernet. Once we need to be portable, start applications in Screen and detach, to leave our application running while we unplug ethernet, and with care, later reconnect and continue; magic indeed!

Software

In addition to various easy to use applications such as raspivid, an acquaintance with a few software languages is helpful to those wishing to develop their own application; some javascript, html5 and svg will be useful and perhaps also C and bash.
When bug hunting, and in case of difficulties, create a reduced test case to test each part independently.

The actual iris tracking software is simple enough, we assume that the iris covers about 30% of the image, and is the darkest part of the image, similar to that illustrated:



Current camera sensors adjust very fast to changes in illumination. A first run assesses the overall tone of each frame. In tests, the upper bound for this darkest 30% ranged from 27 to 200 out of a total possible range of 0-255, one would need to check each frame to evaluate further. We then calculate the centre of this darkness, our supposed iris.

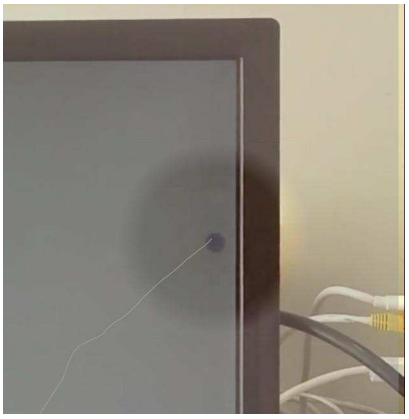
So a few lines of code need to be added to the excellent rpi-mmal camera software from tasanakorn based on his mmal studies:
http://github.com/raspberrypi/userland/tree/master/interface/mmal
Then the Raspberry Pi with eyecam can calculate and display the output in real time at 30fps.

This x,y data is stored in our Redis database for later visualisation, though it may also be sent via ethernet to a website, for realtime interactive gameplay, or via serial uart to our second Raspberry Pi with video camera to mark the user's gaze in real-time.

Uart over serial is used to sync the eyecam and viewcam, as a clapperboard is used to sync audio and video. Drift is bound to arise, there is no editor, nor cuts, and it is almost essential to have a video timestamp, though one is not currently available.

Callibration

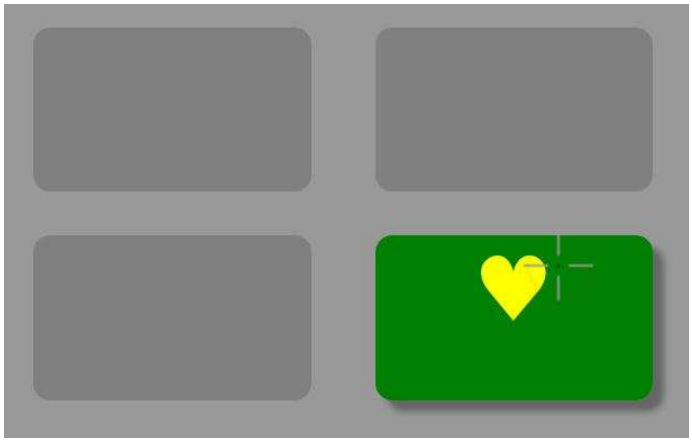
You will need to callibrate the eyecam's eyetracking data with the Raspberry Pi viewcam's sensor's Field Of View. That is, we need to align them in respect of time, and to estimate the user's gaze position on the video frame.



The user should be directly in front of a 1080HD screen which should cover perhaps 20% of their total visual field. The user follows a moving point with their eyes, we use a 4x4 grid which covers the whole screen and is ordered randomly, to prevent the user guessing the next point. This takes about a minute to complete, for the moment assume callibration is complete and successful.

Ready to play my eyetracking game 'Take the Box' and hone your skills.

When designing a game that provides visual feedback one needs to be careful; it can get in the way. The aim of this game is to find which one of 16 boxes contains the hyperlink. The eye moves the cursor in this case a target, which is surprisingly difficult to see, for it moves ever so slightly out of sync.. As you hold your gaze over a box the hue changes slowly from grey to green, and after 2 seconds when a golden heart appears, it's time to try another box.



Should your gaze wander sooner, you start again. Each time you play the game the hyperlink is in a different box, so no cheating.

For now this is a single player html5 game, just plug your new eyecam into your browser box and play. If you've been following, you'll remember the Raspberry Pi cannot run an html5 browser, so for now maybe use a PS3 eyecam and given demand an Raspberry Pi version using x,y coordinates will appear.

For the future

Browser standards are being developed with the natural assumption that a browser is in use. However input devices do not generally require or need a browser. For this reason it may be helpful to develop a range of independent input-device standards and heuristics which whilst not constrained by browser standards, may be considered.

Social media heatmaps that indicate where people are looking during sports events, films, social interaction and other activities.

A suitable cheap configurable eyetracker may provide the affordances that meet medical needs and thus help us better understand the underlying social models of disability.

A results page is under development to help the user evaluate and reject faulty callibration.

The mk2 has the PS3 eyecam in the middle of field of view (FOV), which obstructs the view, but callibration is simple. The Picameral eye-tracker needs a new matrix transformation, which will improve accuracy of the iris centre. So unfortunately, for now the eyetracks illustrated are from the mk2 data, overlaying video from the RPi viewcam.

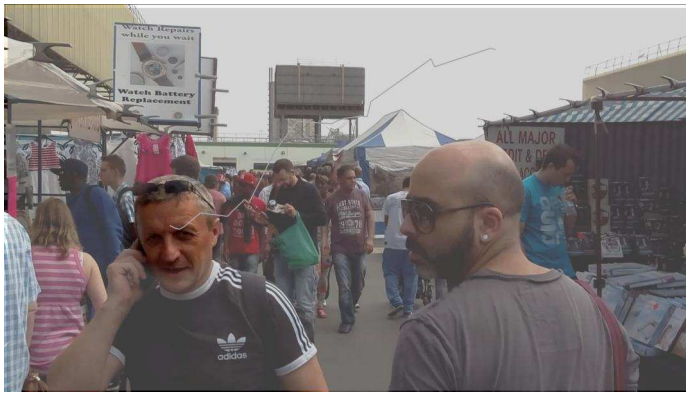
There are many theories of colour and perception. The Ecological approach to visual perception by James J Gibson's describes the sighted animal moving in the natural environment. We follow his belief in this strategy guided by the principal of Gestalt, and the RPi with camera may afford the opportunity to test and develop his hypothesis.

Cartography involves visualising complex data sets, that may include finance, economics, ecology, game theory and is not limited to geography.

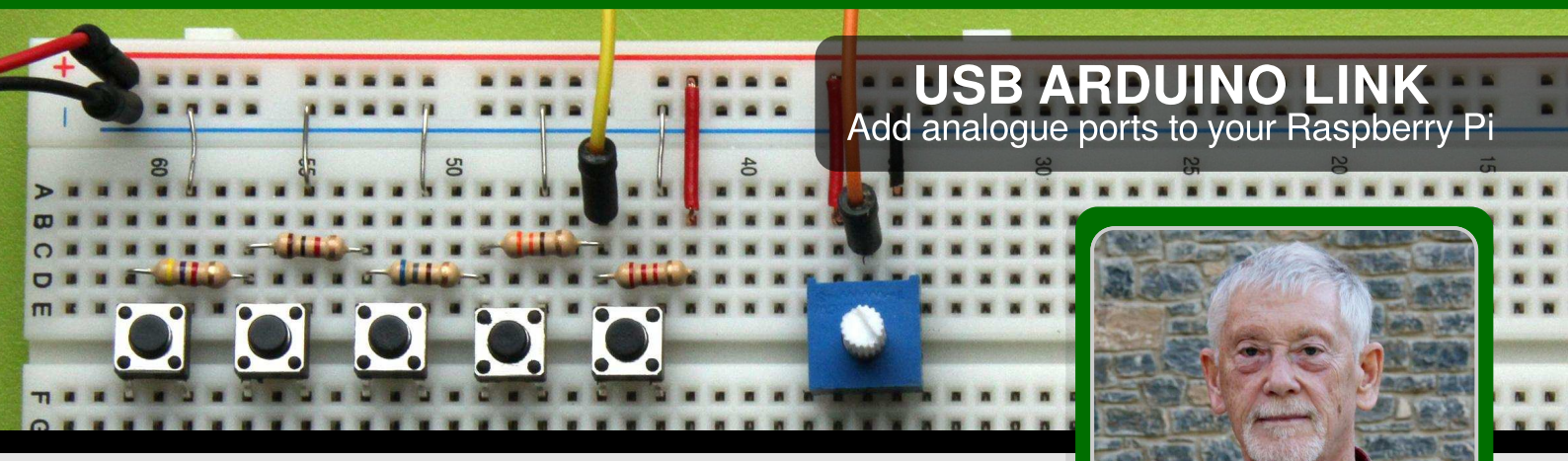
My introductory visualisation is maintained at http://www.peepo.com, where you can play Go, the ancient oriental game of strategy that is played on a 19x19 board. Go is easy to learn but hard to master, and whilst more complex than chess, computers using Monte Carlo Tree Search are making progress.

The eye-tracker described is intended to be used to contrast a study into how different users understand and communicate using maps, with Go as the metaphor, against another study of eye use in vivo. I encourage others to contribute their own ideas.

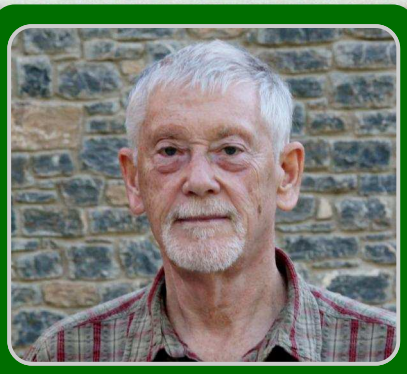
For now the Picameral eye-tracker is a proof of concept.



More details and videos on the website http://www.gnote.org
contact: Jonathan Chetwynd jay@peepo.com



Using a temperature sensor and Arduino analogue pins



Tony Goodhew

Guest Writer

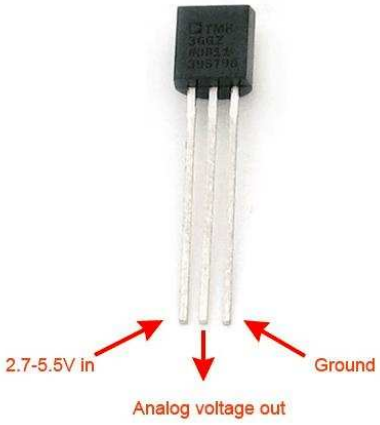
SKILL LEVEL : INTERMEDIATE

USB Arduino link - Part 3

In previous months I have explained how to set up your Raspberry Pi so that it could communicate with an Arduino via a serial USB cable using Nanpy, how to read digital and analogue values from switches and potentiometers attached to the Arduino's pins, and how to write values to these same pins to control LEDS and a Liquid Crystal Display (LCD), all using Python. This month I will explain how to read temperatures and suggest a few ideas for you to try to increase your skills using what has been covered so far.

Reading temperature

A TMP36 is an inexpensive and very easy to use temperature chip which looks like a transistor. If you look at the flat face of the chip, etched with its name, with the legs pointing downwards the left leg goes to 5 volts, the right leg to GND and the centre leg to one of the analogue pins on the Arduino. (See the photograph.) I used A0. Take care to connect it the correct way round after reading the specification at
http://oomlout.com/parts/IC-TMP36-datasheet.pdf
and
http://learn.adafruit.com/tmp36-temperature-sensor
for even more information.



It has a temperature range from -40°C to 125°C with a typical accuracy of ±2oC. It outputs 10mV/°C with a 500 mV offset to allow for negative temperatures. To use the chip you read the analogue value and convert this to a voltage between 0 and 5V. The program carries out the necessary calculations to convert the voltage to a temperature in degrees Celsius.

Due to the variance of the TMP36, the temperature calculated could be up to 2 degrees out in either direction. There could also be small

errors in the analogue to digital converter in the Arduino's chip. If you need a more accurate reading you can calibrate the system. Connect the TMP36 with longer wires. Place the sensor in the corner of a polythene bag, about 8-10 cm long, and close the top tightly with an elastic band to keep out the water. Place several ice cubes in a glass with some water, and immerse the sensor in the water and melting ice. Wait several minutes until the glass, sensor, water and ice settle to the temperature of melting ice. Run the program again. If the results show a consistent 0°C, you are in luck. If not, change the value of 'adjust' in the script until it gives an average voltage of 0.5 volts while the sensor is in the melting ice and water.

```
#!/usr/bin/env python
# TMP36 temperature sensor
# Read values from Analogue pin A0

from nanpy import Arduino
from nanpy import serial_manager
serial_manager.connect('/dev/ttyACM0')
from time import sleep

tmp36 = 14 # Temp Sensor on A0 - Analog input
adjust = 0 # I needed -0.025 here
print"Reading temperature"
for i in range(0,40):
    # Analog input
    val = Arduino.analogRead(tmp36)
    # Convert to voltage
    volts = val * 5.0 /1023 + adjust
    # Calculate temperature
    tempC = volts * 100 - 50
    tempC2 = int(tempC)
    print val, volts, tempC, tempC2
    sleep(0.3)
```

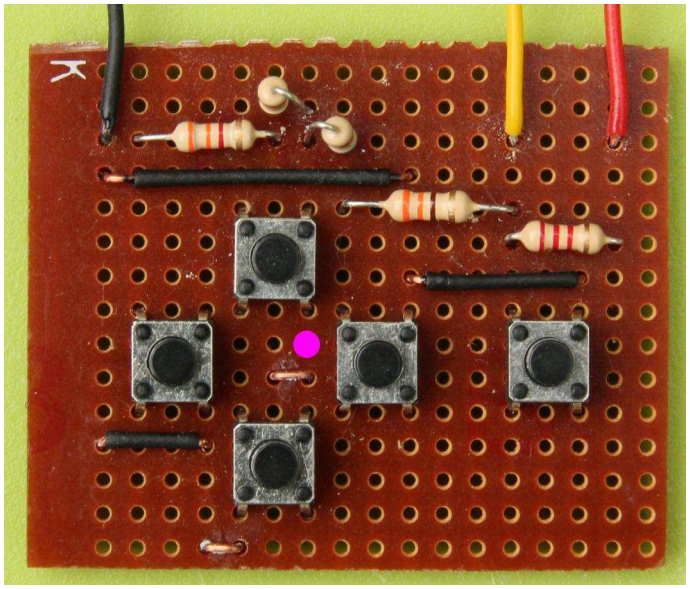
To watch the temperature change you can pinch the sensor with warm fingers or add warm water to the glass if the sensor is still in the plastic bag.

The analogue to digital converter has 10 bit accuracy so each bit will represent 5V /1024 = 0.00488 Volts. This is equivalent to just under ½ °C so the set up can only read accurately to the nearest whole degree C.

Things to try

Extend the multiple buttons circuit with an extra button and resistor. Then change the program to allow for an extra floor.

If you enjoy soldering you could transfer the five buttons and resistors onto a piece strip board – see photograph. Arrange four of the buttons in a diamond pattern to indicate' Up', 'Down', 'Left' and 'Right'. Place the fifth button either in the centre of a large diamond or at the side of a small diamond for 'OK', 'Stop' or 'Fire'. The illustration shows, "one I made earlier". The copper strips are vertical on the back. There are five horizontal connections. Only one track was cut on the underside. The cut track is under the 'Up' button and marked with the magenta blob. This will be useful for the following exercises but the 'in-line' version on breadboard will work as well.



Set up three or more LEDs with resistors (330R or more – we do not need them to be too bright) in a row on digital pins.

a) Light the leftmost LED and make the light appear to move from left to right and back again by pressing two buttons - Right & Left. Use Ctrl-C to stop the program.

b) Control the brightness of one LED by two buttons – Up & Down. Use another button to stop the program. Note: the PWM pins are marked with '~': 3, 5, 6, 9, 10 & 11.
c) Combine the first two ideas so that you can set the individual brightness of all the LEDs with 4 buttons. Use the other button to stop the program.

Change the temperature program so that it changes output from °C to °F while a button is held down. (Move the output to the LCD if you have one.)

Modify the LCD counting program and control the two sets of numbers with the buttons rather than a counted loop. Use one button pair to increase/decrease the left hand number on the display and another pair to control the right number independently. Stop the program if either number reaches 0.

If you do not have an LCD you could write a similar program but output the two numbers in columns on your monitor – one line each time a button is pressed. (Can you keep the tens and units lined up vertically?) Stop the program with Ctrl-C, but trap the error messages.

There is currently a forum on using the Arduino with Nanpy at:
http://www.raspberrypi.org/phpBB3/viewtopic.php?p=368522#p368522

You can use it to ask for help from other users.

Contact me by email via arduinolink@gmail.com with feedback, suggestions or questions.

Thanks again to Andrea Stagi for producing Nanpy.

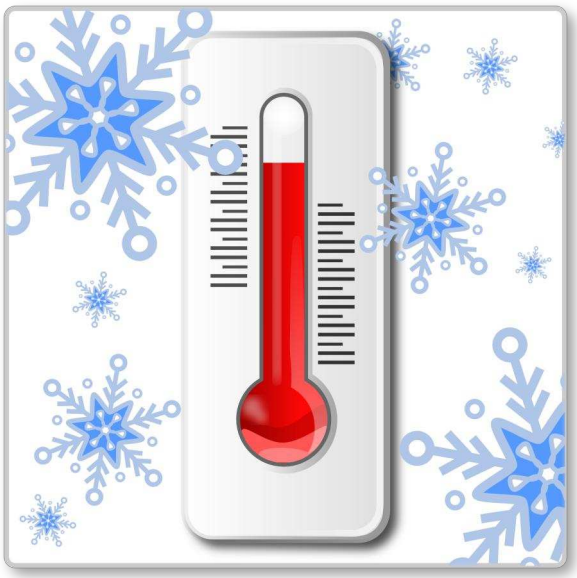
Digital temperature sensors

While it is possible to add analogue temperature sensors to the Raspberry Pi, there are also several digital sensors available. For example, in Issue 5 of the MagPi a Sparkfun tmp102 temperature sensor is connected to a Raspberry Pi over the I2C interface. There is also the possibility of using a sensor with an USB interface such as in the PATOS article in Issue 16 of the MagPi. Lastly, there is the DS18B20 digital temperature sensor available from Adafruit.

The DS18B20 is supplied as a three pin component with a 4.7kOhms resistor or as a prepackaged waterproof unit. The DS18B20 is supported by the LINUX kernel present in the Raspbian distribution. The University of Cambridge have put together some instruction on how to use this temperature sensor:

http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/temperature/

Once the DS18B20 is connected to the appropriate GPIO pins, the temperature sensor will appear as a device under the system bus. The temperature can then be read using cat or using Python to open the device using a simple file open() function call followed by read().



PiGlow

Raspberry Pi® Colour LED Plate
18-channel 8-bit PWM (0-255)
Individually addressable
6 hues + white
~300-500mcd per LED
Fits nicely inside a Pibow

http://shop.pimoroni.com/products/piglow

PIMORONI

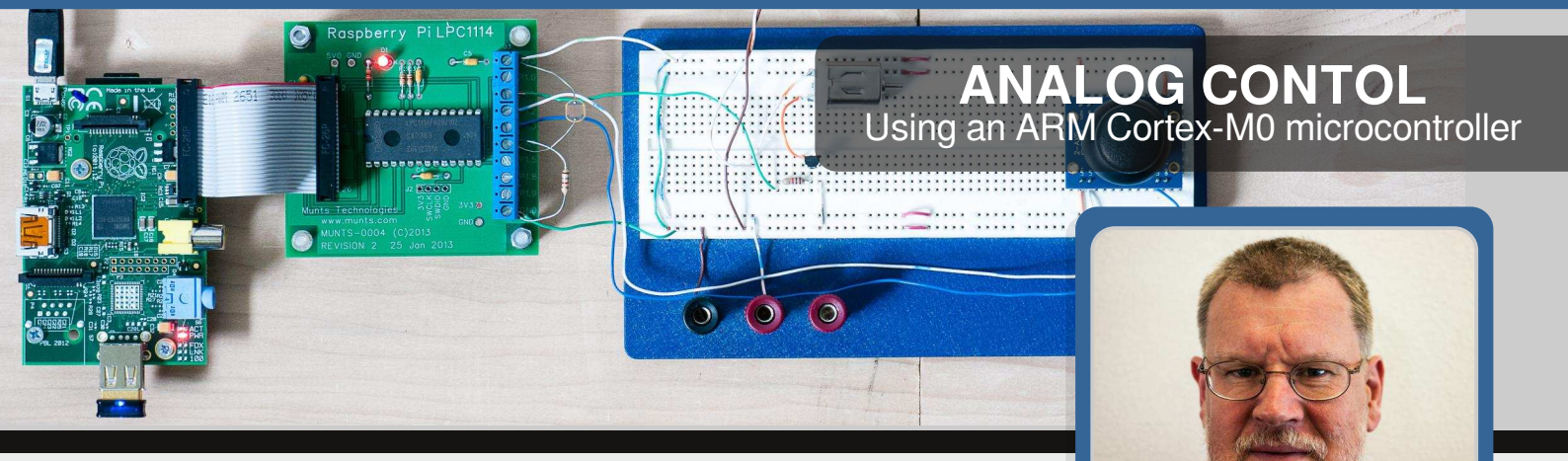
TIMBER

The neat little log cabin for your Raspberry Pi®

Made from real
spruce hardwood

Available From:
http://pibow.com/

PIMORONI



SPI Agent Firmware

SKILL LEVEL : ADVANCED

Philip Munts

Guest Writer

Introduction

In The MagPi Issue 14, I introduced the LPC1114 I/O Processor Expansion board for the Raspberry Pi. In this issue we will examine the factory default SPI Agent Firmware for the LPC1114 and some of the things than can be done with it, particularly with the analog inputs.

During the manufacturing test process, SPI Agent Firmware is written to the LPC1114 microcontroller on each new expansion board. The firmware is used for performing manufacturing tests to ensure the expansion board is fully functional. It can be used for many other interesting projects as well. We will explore how to measure analog signals with C and Python programs, using the LPC1114 I/O Processor expansion board SPI Agent Firmware.

SPI (Serial Peripheral Interface)

SPI (Serial Peripheral Interface) is an industry standard point to point serial communication protocol used between microprocessors and peripheral devices. SPI is capable of medium speed (tens of megahertz) clock rates and is fairly simple to implement. SPI systems are organized in a master/slave fashion, where one device (normally a processor of some type) is designated the master and completely controls the communication process. One or more peripheral devices are designated as slaves (figure 1).

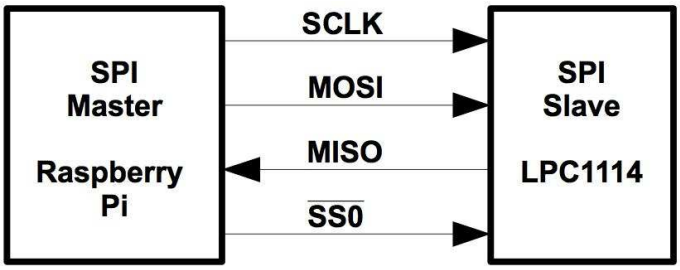


Figure 1 MSPI Connections

There are three I/O signals shared among all peripheral devices on an SPI bus, plus a dedicated chip select signal for each slave device. An SPI bus requires 3+n I/O signals, where n is the number of peripheral devices. The Raspberry Pi P1 expansion connector brings out one SPI bus with two chip select signals, so two different peripheral devices can be controlled by the Raspberry Pi, which is always the master. The LPC1114 microcontroller on the I/O Processor Board is

configured by firmware to act as an SPI slave, using the first chip select signal from the Raspberry Pi.

Since the SPI bus is completely controlled by the master, it does not provide any mechanism for the peripheral device to signal a need for attention. Either the master must continually send a polling message to the slave, or some other signal external to the SPI bus, such as a dedicated interrupt line, must be used to signal the master. The LPC1114 I/O Processor expansion board has two interrupt lines connected between the LPC1114 microcontroller and the Raspberry Pi.

SPI Agent Firmware

I have defined a simple SPI message protocol that the Raspberry Pi uses to send commands to the LPC1114 microcontroller. Each transaction consists of a 3-word (12-byte) command message sent to the LPC1114, a short delay, and a 4-word (16-byte) response received from the LPC1114. See the source code header file include/spi-agent.h in the LPC1114 I/O Processor expansion board source tree for the data structures that implement this protocol. The protocol defines commands for configuring the LPC1114 GPIO pins on the screw terminal block, reading from inputs, writing to outputs, and the like.

Linux provides some system calls for issuing SPI transactions. C programs can call the ioctl() library function with special SPI specific parameters to make these system calls. The ioctl() function is pretty ugly to deal with, so I have written some wrapper functions in common/RASPI_spi.c to make it easier. The functions SPI_open(), SPI_close(), and SPI_transaction() are provided. Most of the C demonstration programs for the LPC1114 I/O Processor expansion board use these three functions for communicating with the LPC1114 microcontroller.

SPI Agent RPC Servers

The SPI wrapper functions provide an efficient way for a C program to issue SPI commands to the LPC1114 microcontroller, but it is not as straightforward for other languages such as Python. By further wrapping the SPI services with an RPC (Remote Procedure Call) server, it is possible to both make them easier to call from Python (or other languages) and make them network accessible.

A RPC client program issues local subroutine calls that are transformed by the RPC library into network messages. These messages are sent to an RPC server program running on the same machine or another machine somewhere else on the network. The RPC server program transforms the incoming messages (using the RPC library again) from the client back into local subroutine calls that perform some service. Then the RPC server sends return values and/or status back to the RPC client. RPC essentially allows you to split a single program into parts that run on different machines.

Two different RPC servers are provided for the LPC11114 I/O Processor expansion board. For this discussion we will use the XML-RPC server. XML-RPC passes data over the network using XML (eXtensible Markup Language) messages over HTTP (HyperText Transfer Protocol). Python has native XML-RPC support, and it is very easy to use. Many other programming languages (especially C, C++, Java, and C#) also have XML-RPC support available.

The following Python 3 program shows how to read a single analog input on the LPC1114 I/O Processor expansion board:

```
import spi_agent
import xmlrpc.client

s = xmlrpc.client.ServerProxy('http://localhost:8080/RPC2')
s.spi.agent.transaction(spi_agent.SPICMD_CONFIGURE_ANALOG_INPUT, spi_agent.LPC1114_GPIO0, 0)
x = s.spi.agent.transaction(spi_agent.SPICMD_GET_ANALOG, spi_agent.LPC1114_GPIO0, 0)[2]
print(str(x))
```

The result, a number between 0 and 1023 inclusive, indicates the magnitude of the analog voltage at terminal GPIO0 / P1.0.

Analog Joystick Demo

For demonstration purposes I connected a Parallax #27800 (Radio Shack #275-030) to the LPC1114 I/O Processor expansion board. Figure 2 shows the connections between the joystick and the expansion board terminal block.

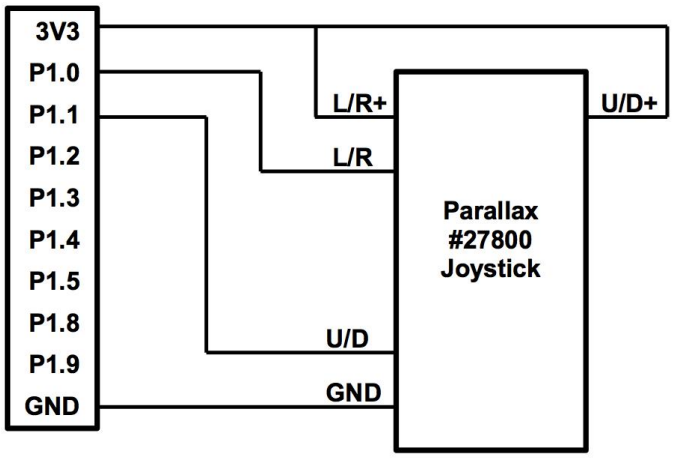


Figure 2 MJoystick Connections

The sample program joystick.py, in the src/adc directory of the LPC1114 I/O Processor expansion board source tree, reads the position of the joystick from the P1.0 and P1.1 analog inputs, and moves a turtle in a turtle graphics window around (Figure 3). Later in another article, we will show how to use the joystick for more sophisticated control purposes.

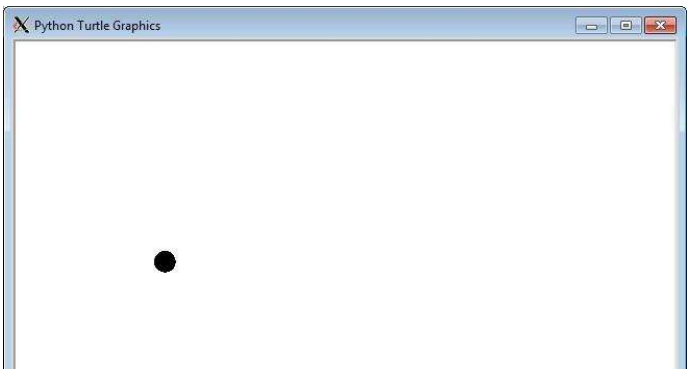


Figure 3 MTurtle Graphics Screenshot

A Raspberry Pi Experiment Board

Figure 4 shows one of several Raspberry Pi experiment boards I have made with the joystick attached. The Raspberry Pi, the LPC1114 I/O Processor expansion board, and a solderless breadboard are mounted on a piece of wood.

The circuit boards are mounted with #4 (3 mm) wood screws and plastic standoffs. The solderless breadboard came with an adhesive pad and is just stuck to the wood.

Links

LPC1114 I/O Processor support info, including how to buy one:
http://munts.com/rpi-lpc1114
XML-RPC information:
http://www.xmlrpc.com
Python 3 XML-RPC client library documentation:
http://docs.python.org/3.2/library/xmlrpc.client.html
Parallax 2-axis joystick information:
http://learn.parallax.com/KickStart/27800

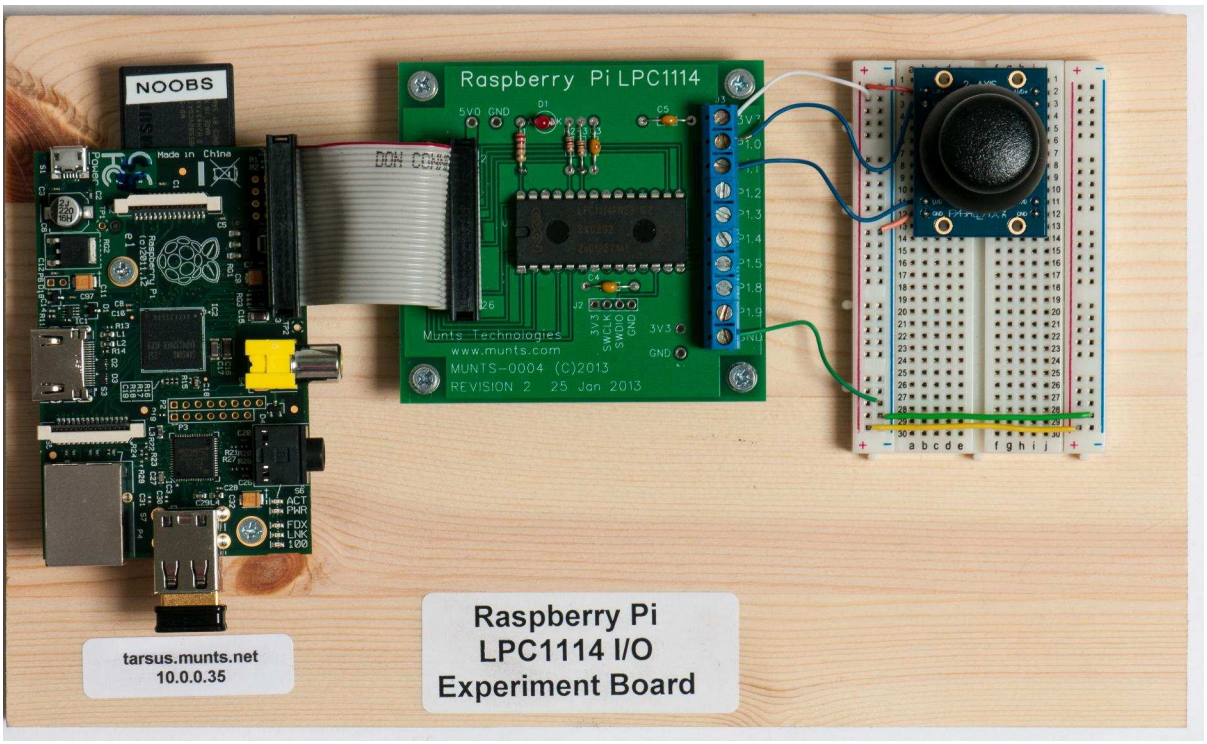


Figure 4 – ‘ d‹ sehɯ_‖ –‖ FõõõO ¯shu″ hq› Er dʊg

Finest
Raspberry Pi Accessories
phenoptix
available from
phenoptix.com
est.2003

VISIT US ON

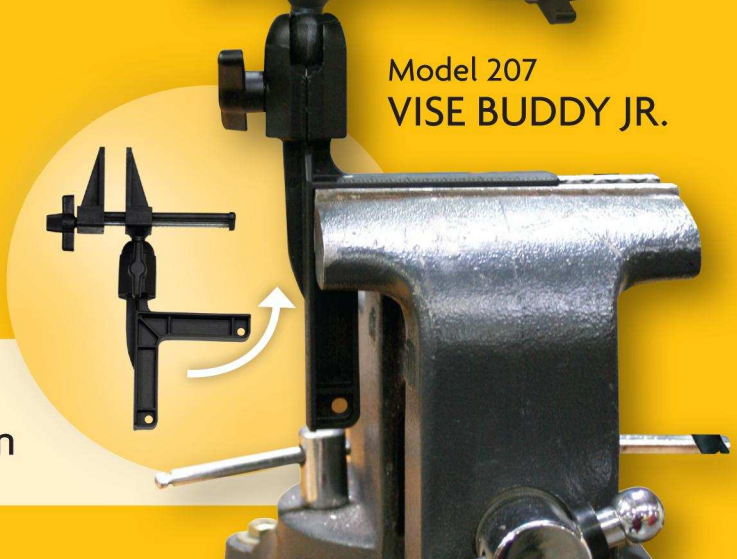
WORLD'S MOST VERSATILE CIRCUIT BOARD HOLDERS

Model 209
VACUUM
BASE PV JR.

Model 201
PV JR.

- Work-holding tools for electronics projects
- Circuit board holders make soldering easy & fun
- Versatile hobby vises for any project

Model 207
VISE BUDDY JR.



PANAVISE®
Innovative Holding Solutions

www.panavise.com

PIMORONI

PANAVISE IS AVAILABLE AT
http://shop.pimoroni.com

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

IO Pi

32 digital input/output channels for your Raspberry Pi. Stack up to four IO Pi boards to give you 128 I/O channels.

£16.99

RTC Pi

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

£9.75

ADC Pi

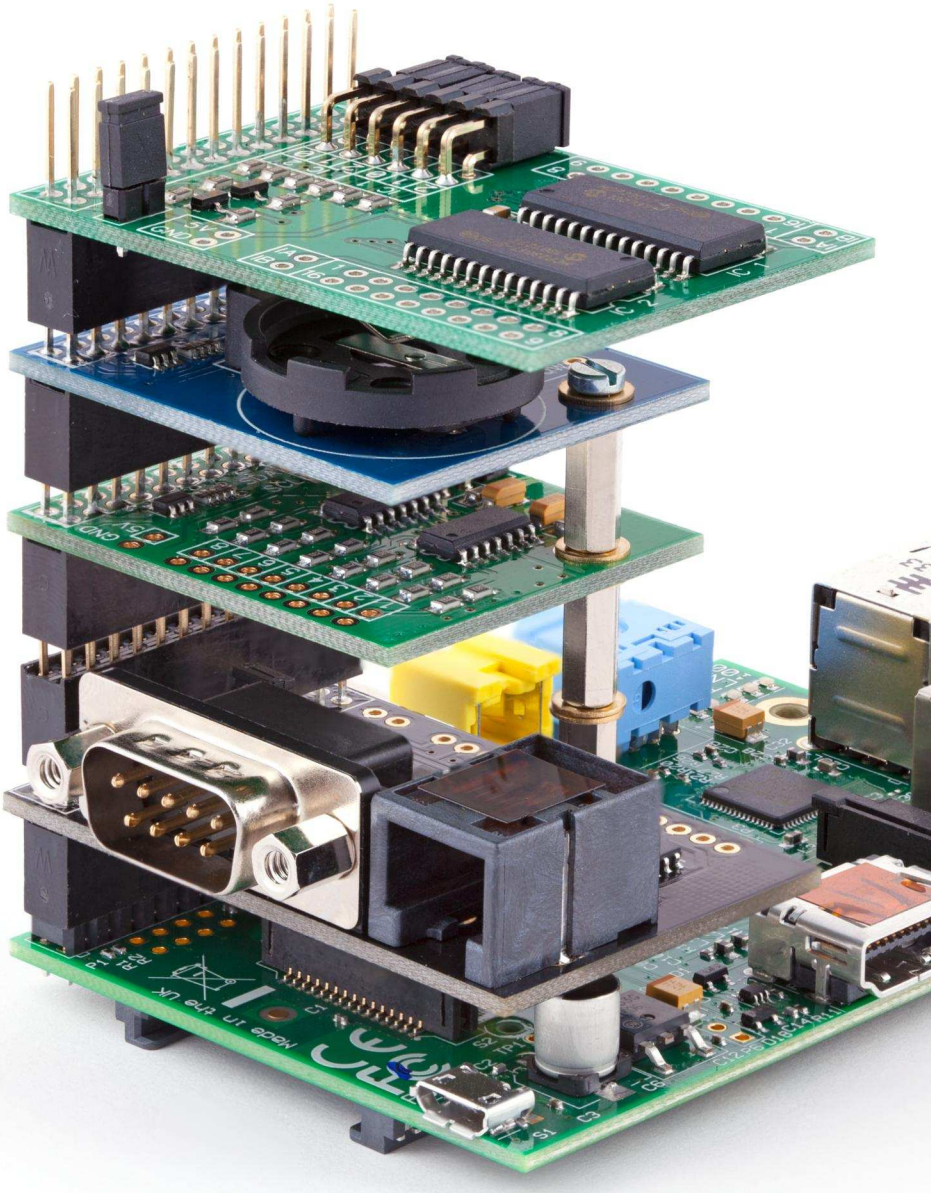
8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels on your Raspberry Pi.

£17.99

Com Pi

RS232 and 1-Wire® expansion board adds a serial port to your Raspberry Pi. Ideal for the Model A to enable headless communication.

£19.99



AB electronics UK

www.abelectronics.co.uk

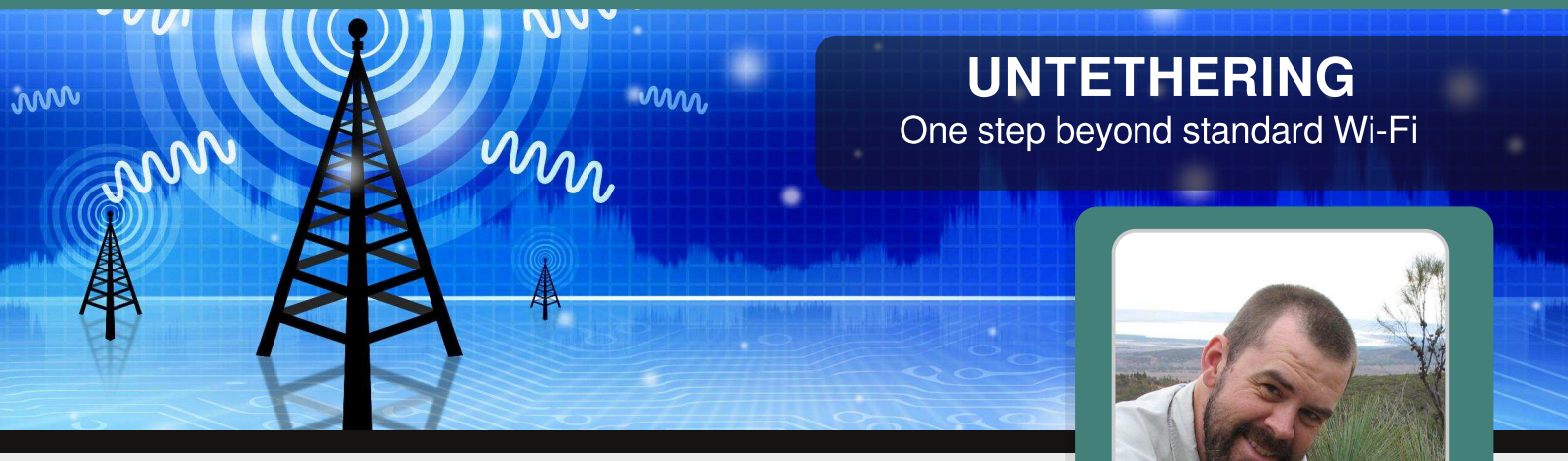
Win a selection of Raspberry Pi expansion boards from AB Electronics UK

Enter our competition and you could win a selection of Raspberry Pi expansion boards including an IO Pi, RTC Pi, ADC Pi and Com Pi.

Visit our website for more information.

For your chance to win visit

www.abelectronics.co.uk/magpi/



Long-range wireless with the RPi900

Matthew Hollingworth
Guest Writer

SKILL LEVEL: ADVANCED

We all know the benefits of Raspberry Pi for hardware projects: easy to program, low power, and cheap almost to the point of disposable. Add a Wi-Fi dongle and you can put one anywhere around the house. But how about further afield? I wanted to put a Raspberry Pi on a local hilltop and still be able to log in. This would be great for weather stations, webcams and a host of other remote telemetry projects.

A good solution for this sort of long-range communication is to use a "radio module", a plug-in circuit which communicates wirelessly with others in its network. These modules typically use the 900 MHz or 2.4 GHz radio bands, which are available for license-free use in many countries.

I chose the DNT900 module from RFM. At $70 it's very affordable, and can work at extremely long range up to 40 miles line-of-sight, RFM claims. (There's also a compatible 2.4 GHz module for use in Europe, where 900 MHz is restricted.) The radios can be connected in pairs, or in a more complex network. You can also operate a radio as a repeater to extend range. The radios even include analog and digital I/O for controlling things and making simple measurements.

I designed a prototype circuit board which holds the DNT900 radio module and connects to the Raspberry Pi's GPIO header. The serial port (UART) pins allows the Raspberry Pi to exchange data with with radio. I included a 5V power supply, sufficient for both the Raspberry Pi and DNT900 radio and able to be supplied directly from a 12 V solar power system.

One feature I wanted was for the remote radio to be able to control power to the remote Raspberry Pi. This would be very useful for turning the Raspberry Pi off when not needed, saving battery power. Later, I also added a real-time clock, a useful feature which allows the Raspberry Pi to keep time without a network

connection.

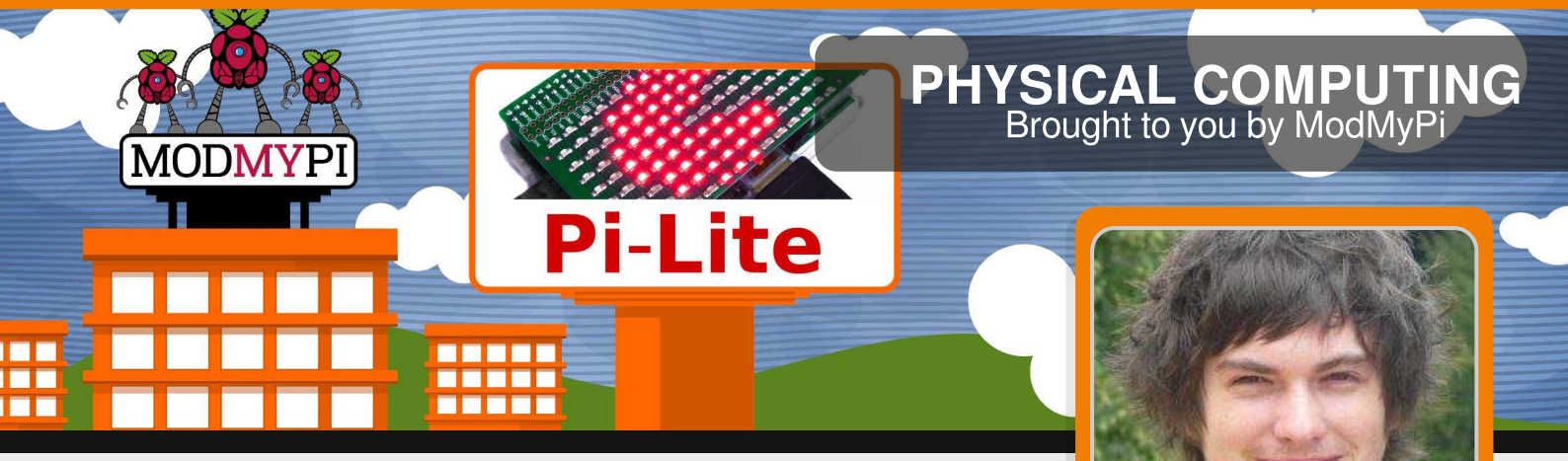
Once I had a working prototype, my attention turned to software. With one Raspberry Pi acting as a base station, how would I configure the radio and exchange data with the remote Raspberry Pi? The DNT900 radio has a special command protocol for doing just this. You can manually construct these commands and sent them to the Raspberry Pi serial port, but this quickly becomes tedious. I decided to do it the right way: create a Linux driver.

In this case the driver turned out to be a "line discipline". This is an obscure bit of kernel code which acts as a "translator" for the serial port. Basically, it implements the DNT900 command protocol so you don't have to. Writing the code was a big project in its own right, and the end results were great. When the driver is loaded, all the configuration registers for the radio are presented as attribute files in the sysfs file system, so changing them is easy. For every radio which is connected, a "virtual TTY" is presented, meaning that sending data to a remote Raspberry Pi is as simple as writing to its corresponding TTY device file.

What's nice about the TTY interface is that I can use it to set up a full internet connection for the remote Raspberry Pi (albeit a slow one). For this I use Point-to-Point Protocol, the same as for dialup internet. Just a few extra steps to set this up, and I can SSH into the remote Raspberry Pi using only the radio link. This is great for maintenance, since I can reconfigure software on the Raspberry Pi without needing physical access to it. Handy for a remote location!

For my base station, I boxed up one of my boards and a Raspberry Pi and bolted in to my roof, along with a solar panel, battery and antenna. I have yet to test the full range of the radio, but I can certainly reach the local hill!

Working on this project is great fun. I've documented my design at http://rpi900.com. The site includes technical tutorials on setting everything up as well as tips on choosing an enclosure, antenna and solar power system for your remote Raspberry Pi. RPi900 boards are also available to purchase from the site.



Buttons and switches with the Raspberry Pi

Jacob Marsh

ModMyPi

SKILL LEVEL : BEGINNER

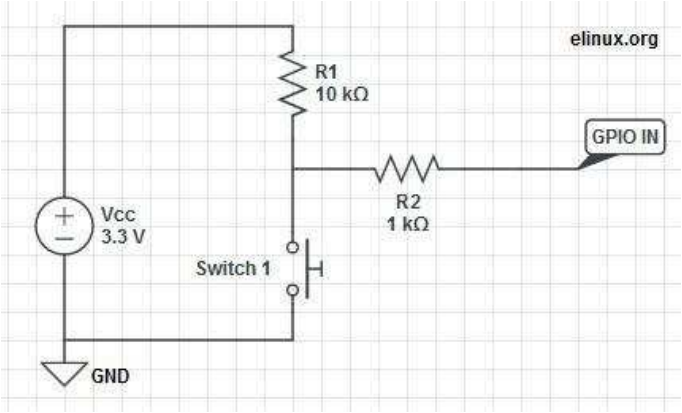
Buttons and switches are a fundamental part of 'physical' computing. This beginner's tutorial is designed to teach the basics of physical operation and programming with the Raspberry Pi using a simple momentary switch setup.

The tutorial requires a few simple components that are available from ModMyPi (product codes in brackets):

- Medium Breadboard (BB2) – For laying out our components & circuit.
- Male to Female Jumper Wires (JW8) – For jumping between the RPi & breadboard.
- PCB Mount Switch (TAC001) – A four point basic momentary switch.
- ModMyPi's Ridiculous Resistor Kit (RK995) – to protect your Pi & calibrate the float voltage.
- 10KΩ Resistor - (Brown, Black, Black, Red, Brown)
- 1KΩ Resistor - (Brown, Black, Black, Brown, Brown)
- Breadboard Jumper Wire Kit (140KI) – For easy jumping on the breadboard.

The circuit

The purpose of this circuit is to enable the Raspberry Pi to detect a change in voltage and run a program when the button (Switch 1) is pressed. This requires three GPIO pins on our Raspberry Pi: the first will provide a signal voltage of 3.3V (Vcc), the next will ground the circuit (GND), and the third will be configured as an input (GPIO IN) to detect the voltage change.



When a GPIO pin is set to input, it doesn't provide any power and consequently has no distinct voltage level; defined as 'floating'. We need the pin to be capable of judging the difference between a high and low voltage, however in a floating state it's liable to incorrectly detect states due to electrical noise. To enable the pin to see the difference between a high or low signal we must 'tie' that pin, calibrating it to a defined value; 3.3V in this case!

To tie the input pin, we connect it to the Vcc 3.3V pin, hence when Switch 1 is open, the current flows through GPIO IN and reads high. When Switch 1 is closed, we short the circuit and the current is pulled to GND; the input has 0V, and reads low! The large R1 (10kΩ) resistor in this circuit ensures that only a

little current is drawn when the switch is pressed. If we don't use this resistor, we are essentially connecting Vcc directly to GND, which would allow a large current to flow, potentially damaging the Pi! To make the circuit even safer in case we get something wrong, we add the R2 (1kΩ) resistor to limit the current to and from GPIO IN.

The switch

Four point switches are wired in a very similar manner to two point switches. They're simply more versatile, as you can have multiple isolated inputs into the same switching point. Checking the diagrams, Pins 1 & 2 are always connected, as are Pins 3 & 4. However, both pin pairs are disconnected from each other when the button is not pressed e.g. Pins 1 & 2 are isolated from Pins 3 & 4. When the button is pressed the two sides of the switch are linked and Pins 1, 2, 3 & 4 are all connected!

In 'momentary' switches the circuit disconnects when pressure is removed from the button, as opposed to 'toggle' switches when one push connects and the next push disconnects the internal switch circuit.

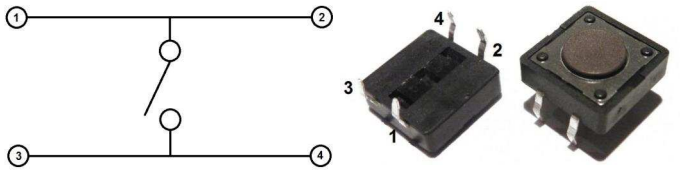
Where does it all go?

WARNING. When hooking up to GPIO points on your Raspberry Pi care must be taken, as connecting the wrong points could permanently fry your Pi. Please use a GPIO cheat-sheet, and double check everything before switching it on. I will denote each GPIO point by its name, and physical location, for example GPIO P17 is actually located at Pin 11, denoted: GPIO P17 [Pin 11]. The irregularities are a result of the pin names being referenced by the on board chip rather than their physical location.

1. Connect Pi to Ground Rail. Use a black jumper wire to connect GPIO GND [Pin 6] on the Pi to the Negative rail on the breadboard – the rail on the edge of the board with the negative sign (-).

2. Connect Pi 3.3V to Positive Rail. Use a red jumper wire to connect GPIO 3.3V [Pin 1] on the Pi to the Positive rail on the breadboard – the edge rail with the positive sign (+).

3. Plug your switch in. When breadboarding, make sure all of the legs are in separate rows. To achieve this straddle the central channel on the breadboard.

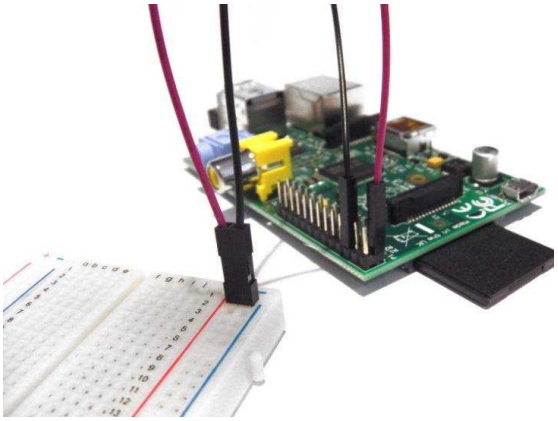


4. Add 10kΩ Resistor. Connect this from Switch Pin 1, to the positive (+) rail of the breadboard. Orientation of standard film resistors is unimportant.

5. Connect Switch to Ground. Use a breadboard jumper wire to hook Switch Pin 3 to the ground (-) rail.

6. Connect Switch to 1kΩ Resistor. Add this resistor between Switch Pin 1 and the 10kΩ Resistor and take it to a clear rail.

7. Connect Switch to Signal Port. We'll be using GPIO P17 to detect the 3.3V signal when the switch is pressed. Simply hook up a jumper between GPIO P17 [Pin 11] on the Pi and the 1kΩ Resistor rail.



That's our circuit built! Next time, we'll write a simple program in Python to run when we press the switch!

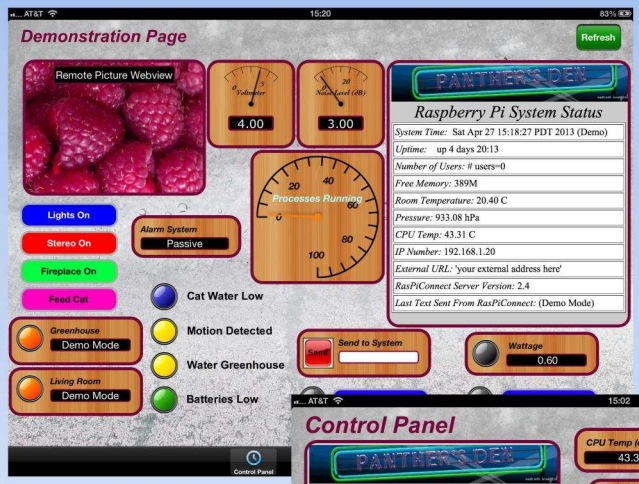
MODMYPI

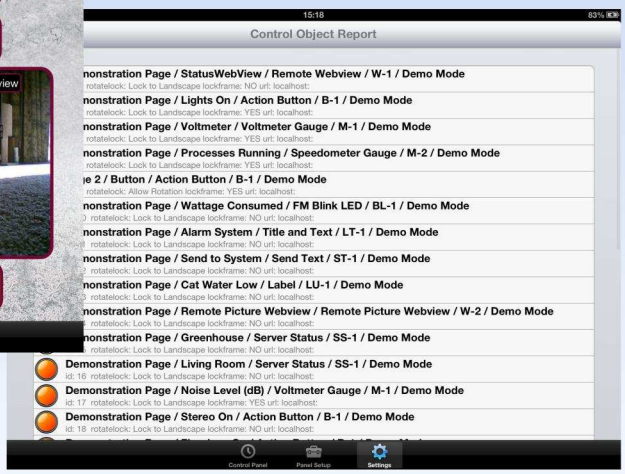
This article is sponsored by ModMyPi

All breakout boards and accessories used in this tutorial are available for worldwide shipping from the ModMyPi webshop at www.modmypi.com

RasPiConnect

Connect your Raspberry Pi to the World!




Control Object Report
Demonstration Page / Lights On / Action Button / B-1 / Demo Mode
Demonstration Page / Voltmeter / Voltmeter Gauge / M-1 / Demo Mode
Demonstration Page / Processes Running / Speedometer Gauge / M-2 / Demo Mode
Demonstration Page / Wattage Consumed / FM Blink LED / BL-1 / Demo Mode
Demonstration Page / Alarm System / Title and Text / LT-1 / Demo Mode
Demonstration Page / Send to System / Send Text / ST-1 / Demo Mode
Demonstration Page / Cat Water Low / Label / LU-1 / Demo Mode
Demonstration Page / Remote Picture Webview / Remote Picture Webview / W-2 / Demo Mode
Demonstration Page / Greenhouse / Server Status / SS-1 / Demo Mode
Demonstration Page / Living Room / Server Status / SS-1 / Demo Mode
Demonstration Page / Noise Level (dB) / Voltmeter Gauge / M-1 / Demo Mode
Demonstration Page / Stereo On / Action Button / B-1 / Demo Mode
Control Panel
Panel Setup
Settings

Now Supports Arduino with in-app purchase

Available on the App Store

RasPiConnect connects your Raspberry Pi to the outside world. It allows you to control virtually anything you connect to your Raspberry Pi from your iPad or iPhone.

- EASY to setup - no syncing required
- Buttons, gauges, webpages, webcam pictures and more!
- Build your pages on your iPad/iPhone
- Supports multiple Raspberry Pis
- Five pages of control panels
- Unlimited Controls
- Exchange your panels with friends
- Supports any computer that supports Python (windows, linux, etc.)
- Now allows custom backgrounds

MiloCreek
Where Innovation Flows

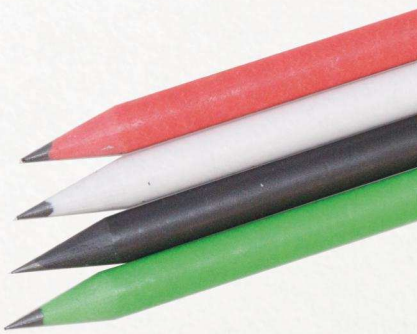
The Raspberry Pi is helping millions of kids write their first

print 'hello world'

We'd like to thank you for
making cool projects,
spreading the word,

...and also for buying sweet, sweet swag

Your generous support helps us do more*
http://swag.raspberrypi.org



AUTHENTIC SWAG EDUCATING SINCE '12

*Hello World is alright, but we've got to teach kids "20 GOTO 10" as well

XML for the Raspberry Pi: Part 3

John Shovic

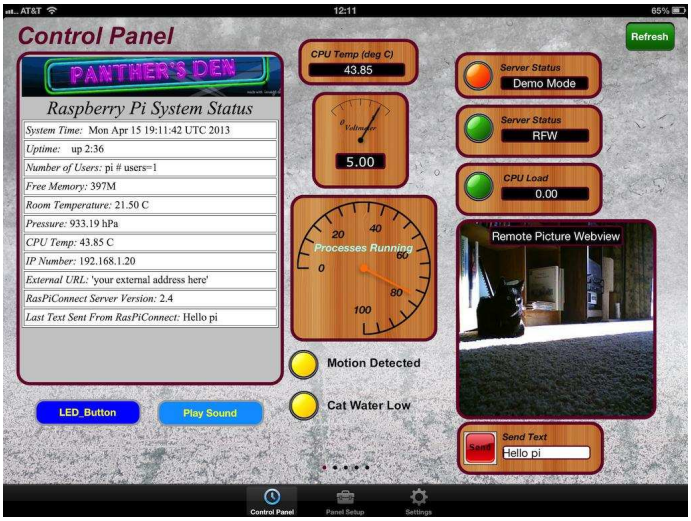
Guest Writer

SKILL LEVEL : INTERMEDIATE

Introduction

This series of articles discusses the use of XML on applications for the Raspberry Pi. Part One covered what is XML and the format of the data structures, and Part Two covered building and parsing XML in Python. In Part Three we will see how XML is used as a client / server application, RasPiConnect. RasPiConnect is an iPad/iPhone app that connects and displays information for any number of Raspberry Pi's via a defined XML interface.

The Application



Our example application consists of a client and server. The client is an Apple App running under iOS written in Objective C and the server is a Raspberry Pi server (the Server) program written in Python. The purpose of the RasPiConnect App (the Client) is to display information and control your Raspberry Pi from an iPad or iPhone device. The Python code mentioned in this article can be found at https://github.com/milocreek.

What is XML being used for in this program?

XML is being used for three purposes in this program. 1) For the communications channel (over HTTP) from the App to the Raspberry Pi. 2) For the communications channel from the Raspberry Pi to the App (over HTTP) and 3) For persistent program and screen configuration storage in the App.

The communications channels

The Client communicates with the Server by sending and receiving HTTP. The Objective C code on the Apple iOS device for sending and receiving is beyond the scope of this article. The Python code on the Server for receiving and sending XML code is very straight forward. In the Server we use the web.py library from webpy.org. This is a light weight webserver program readily available by running the following command on your Raspberry Pi.

```
sudo apt-get install python-webpy
```

There is a bi-directional communication channel between the Server and Client. Both directions are handled by HTTP requests. The Client sends requests for data and action with all the requests grouped together. The XML sent from the Client to the Server looks like this:

```
<XMLObjectXMLRequests>
 <XMLCOMMAND>
  <OBJECTSERVERID>LT-1</OBJECTSERVERID>
  <OBJECTNAME>CPU Text and Label
  </OBJECTNAME>
  <OBJECTTYPE>1</OBJECTTYPE>
   ...
 </XMLCOMMAND>
  ...
 </XMLCOMMAND>
</XMLObjectXMLRequests>
```

This XML contains multiple requests to the Server for retrieving information and sending action requests to the Server. Note the multiple <XMLCOMMAND> entities in the structure.

The returning XML from the Client looks very similar.

```
<XMLRESPONSES>
 <XMLCOMMAND>
  ...
 </XMLCOMMAND>
 <XMLCOMMAND>
  <OBJECTSERVERID>LT-1</OBJECTSERVERID>
  <RESPONSE>
   <![CDATA[43.31, 43.31, CPU Temp (deg
C)]]>
  </RESPONSE>
  ...
 </XMLCOMMAND>
</XMLRESPONSES>
```

Parsing the XML

Parsing this XML into the individual entities (<XMLCOMMANDS> above), is a simple use of the ElementTree Python library as shown previously in Part Two of this article. Once the requests have been parsed and validated, the server executes the requests one at a time, while building a new XML structure containing the responses to the commands. The <XMLResponses> structure is then sent to the Client using one HTTP connection rather than multiple connections.

```
...
class RasPi:
    def POST(self):
        web.header('Content-Type',
'text/html')
        incomingXML = web.data()
        root = ET.fromstring(incomingXML)
        # iterate through all the values
        for element in
root.findall('XMLCOMMAND'):
...
```

Building XML to send to the Client

Building the XML to be sent back via the HTTP request from the Client to the Server is done by constructing a string of concatenated XML commands and then returning the string as part of the web.py POST HTTP request.

```
  # start of building the XML responses
  outgoingData="<XMLRESPONSES>"
    ...
    outgoingData +="<XMLCOMMAND>"
     ...
     outgoingData +="</OBJECTTYPE>"
     outgoingData +="<OBJECTID>"
     outgoingData += "%i" % objectID
     outgoingData +="</OBJECTID>"
     # done with FOR loop
  outgoingData+="</XMLRESPONSES>"
  return outgoingData
```

Receiving and sending the XML

In web.py, the incoming XML is placed in a string as above and then parsed. The responses are sent back to the client from the POST function by returning a string.

Conclusion

XML is a very useful means for storing and transmitting data across disparate computer systems. It is usable by large and small computers alike. To learn more about using XML on a Python based platform try the following websites:

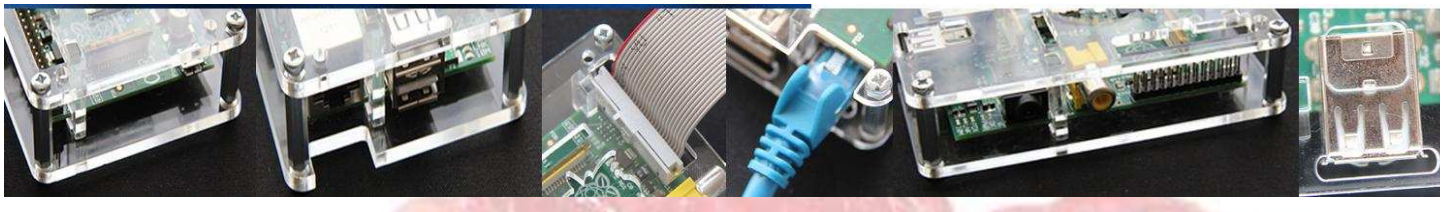
http://docs.python.org/2/library/xml.etree.elementtree.html

http://eli.thegreenplace.net/2012/03/15/processing-xml-in-python-with-elementtree

and a tutorial video on elementtree:

http://www.youtube.com/watch?v=LNYoFo1sdwg

OCTOBER COMPETITION



Once again The MagPi and PC Supplies Limited are proud to announce yet another chance to win some fantastic Raspberry Pi goodies**!**

This month there is one MASSIVE prize!

The winner will receive a new Raspberry Pi 512MB Model B, an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires!

For a chance to take part in this month's competition visit:
http://www.pcslshop.com/info/magpi

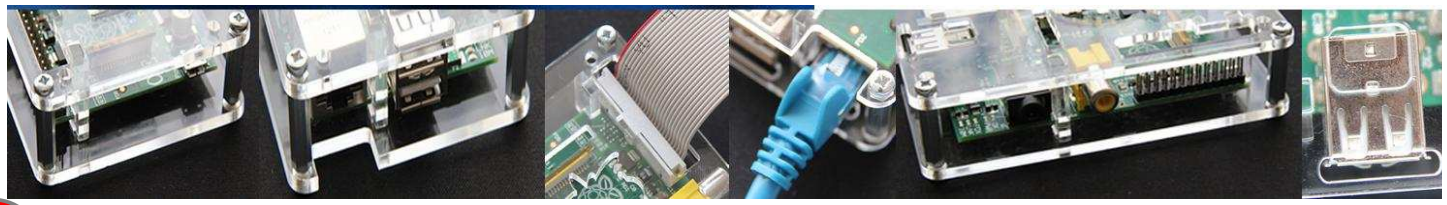
Closing date is 20th October 2013. Winners will be notified in next month's magazine and by email. Good luck!

To see the large range of PCSL brand Raspberry Pi accessories visit
http://www.pcslshop.com

August's Winner!

The winner of a new 512MB Raspberry Pi Model B plus an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires is Russ Lovering of Plymouth!

Congratulations. We will be emailing you soon with details of how to claim your prizes!



The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Creative Coding on the Raspberry Pi with openFrameworks

When: Starts Friday 11th October 2013, 6.00pm and continues all weekend
Where: Frank-Ratchye STUDIO for Creative Inquiry, Pittsburgh, PA, USA

A weekend of teaching, learning, hacking and making with the Raspberry Pi and openFrameworks.
Tickets from: http://eventbrite.com/event/8173616507

Exeter Raspberry Jam

When: Saturday 12th October 2013, 10.00am until 12.00pm
Where: Hoskins Room, Exeter Central Library, Castle Street, Exeter, EX4 3PQ, UK

Featuring displays of projects and robotics, use of GPIO and helpdesk. Bring your own project or come and experience the Raspberry Pi for the first time. All welcome. http://goo.gl/EjxJDF

Leicester Raspberry Jam

When: Saturday 26th October 2013, 12.00pm until 2.00pm
Where: Phoenix Digital Arts Centre, 4 Midland Street, Leicester, LE1 1TG, UK

Bringing together Raspberry Pi computers, projects and a demonstration of a 3D printer.
www.facebook.com/LeicesterDigitalMakers and www.eventbrite.com/event/7286410851

Stevenage Raspberry Jam

When: Saturday 26th October, 10.30am until 1.00pm
Where: North Herfordshire College, STEM Centre, Monkswood Way, Stevenage, SG1 1LA, UK

Featuring a showcase area for all things Raspberry Pi.
Tickets from: stvraspberryjam-eorg.eventbrite.co.uk

Learning Computer Science with the Raspberry Pi

When: From Wednesday October 30th at 12.00pm continuing to November 2nd
Where: The LAB Miami, 400 Northwest 26th Street, Miami, FL 33127, USA

Programming for Non Programmers: October 30th - adults; October 31st - educators. Learn prgramming rapidly: November 2nd - young people. Further information: http://labraspberrypiworkshops-es2.eventbrite.com

MEMORY MANAGEMENT
union, malloc() & free()

W. H. Bell

MagPi Writer

7 - Unions & dynamic allocation

SKILL LEVEL : ADVANCED

Welcome back to the C Cave. How did you get on with the challenge problem given in Issue 13?

Challenge solution

```
#ifndef PYTHIA_H
#define PYTHIA_H

typedef struct {
  int msel;
  int mselpd;
  int msub[500];
  int kfin[81][2];
  double ckin[200];
} PYSUBS;
extern PYSUBS pysubs_;

void pyinit_(char *, char *, char *, double *, int, int, int);
void pystat_(int *);
void pyevnt_(void);
void pylist_(int *);
void pyhepc_(int *);

#endif
```

To use the solution, write it to a file called `python.h` and put it into the same directory as the C program, make file and Pythia FORTRAN file. Then compile and run the program.

Unions

The syntax of declaring a `union` is the same as that of a `struct`. However, unlike a `struct`, each of the data members share the same memory space as each other. The amount of memory allocated to an `union` instance is set by the largest data member. Similar to `structs`, `union` data members can be `structs`, `unions` or simple types (`float`, `int`, etc.). The memory allocation can be demonstrated with a simple example,

```
#include <stdio.h>

union share {
  int index;
  float value;
  unsigned char bytes[8];
};

int main() {
  union share s;
  printf("Size of s = %lu\n",sizeof(s));
  printf("Size of s.index = %lu, address of s.index = %p\n",sizeof(s.index),&s.index);
  printf("Size of s.value = %lu, address of s.value = %p\n",sizeof(s.value),&s.value);
  printf("Size of s.bytes = %lu, address of s.bytes = %p\n",sizeof(s.bytes),&s.bytes);
  return 0;
}
```

where the result of compiling and running this code should be similar to:

```
Size of s = 8
Size of s.index = 4, address of s.index = 0x7fff5fbff040
Size of s.value = 4, address of s.value = 0x7fff5fbff040
Size of s.bytes = 8, address of s.bytes = 0x7fff5fbff040
```

The `unsigned char` type uses one byte per element in the array. This means that the array uses eight bytes. The `int` and `float` types typically use four bytes with the default gcc compiler options. Therefore, an instance of the `union` uses eight bytes of memory. Notice that the memory address of each of the data members is the same. Try modifying this example by exchanging the `union` for `struct`. Then re-run it and compare the size and addresses of the data members. The result will be similar to:

```
Size of s = 16
Size of s.index = 4, address of s.index = 0x7fff5fbff030
Size of s.value = 4, address of s.value = 0x7fff5fbff034
Size of s.bytes = 8, address of s.bytes = 0x7fff5fbff038
```

which demonstrates that the memory allocation of a struct is sequential and the total size is the sum of the sizes of the data members.

An `union` can be useful for encoding or decoding variables in a binary data stream. For example, one of the `union` members could be a basic type such as an `unsigned char` array. Then the basic type is written or read from a binary file, while the more complex type is used within the program.

Dynamic memory allocation

Sometimes the size of the memory needed for a given stage of a program is not known at compile time. Instead of declaring a large block of memory when it might not be needed, dynamic memory allocation allows a program to request or free memory use as needed.

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
  int *buffer = 0;
  unsigned int i, num_elements = 0;

  if(argc != 2) {
    printf("Usage: %s <number of elements>\n",argv[0]);
    return 1;
  }
```

```
  sscanf(argv[1],"%u",&num_elements);
  if(num_elements == 0) {
    printf("Error: the size of the array must be greater than 0\n");
    return 2;
  }

  /* Sleep for 10 seconds. */
  sleep(10);

  /* Request a block of memory long enough for 1000 integers */
  buffer = (int*)malloc(sizeof(int)*num_elements);

  /* Check if the request was successful or not. */
  if(!buffer) {
    printf("Error: malloc failed\n");
    return 3;
  }

  /* Now use the memory as needed. */
  for(i; i<num_elements; i++) {
    buffer[i] = i;
  }

  /* Print the last element to show that it is used. */
  printf("buffer[%u]=%u\n",num_elements-1,buffer[num_elements-1]);

  /* Sleep for 10 seconds. */
  sleep(10);

  /* Free the memory allocated to buffer. */
  free(buffer);

  /* Sleep for 10 more seconds. */
  sleep(10);

  return 0;
}
```

This program demonstrates how to request a block of memory on the heap and then free it again. Compile the program using gcc, then open another terminal window and type `top`. In the first terminal window, run the program

```
./dynamic-allocation 10000000
```

and then watch what happens using `top`. The memory allocation of the program is initially quite small. Then when the array is declared, the size will jump up. Then after another ten seconds, the size of the example program will drop down again.

The function `malloc` returns a void pointer and the function `free` takes a void pointer as the single input variable. Since `malloc` returns a void pointer, the example program casts the point pointer to the type of pointer needed. If the request for more memory fails, then `malloc` returns null which is zero as an integer or false as a boolean expression. Null could be returned if there is a memory leak in the program that uses up all of the available memory or if the computer has no more memory available. A memory leak is caused when memory is requested, but is not freed. If the leak is inside a loop, the amount of memory the program uses can increase quickly. Unlike variables on the stack (variables defined non-dynamically), the variables on the heap are not automatically cleaned up. The heap is outside of the scope of a given function and must be explicity cleaned up with `free`. When `free` is called, the memory is released back to the system but the pointer is not set to be null. Therefore, if `malloc` and `free` are used within a loop the pointer should be set to null afterwards, to allow a test on the pointer after the next `malloc` call.

Dynamic memory allocation memory functions in C are declared in the `stdlib.h` header file. Type `man malloc` for more information on the dynamic memory functions.

Scope

A variable is said to go out of scope when the function it is declared in finishes. For example,

```
void fun() {
  int i = 0;
}
```

the variable `i` goes out of scope at the end of the function `fun`. This means that the variable `i` is no longer allocated memory and cannot be used by an external program. Do not try to access the memory allocation of `i` outside the function fun. Attempts of the form

```
include <stdio.h>

int* fun() {
  int i=0;
  return &i; /* Return the memory address of i.  Do not do this! */
}

int main() {
  printf("%p\n",fun()); /* Print the memory address of i within fun() */
  return 0;
}
```

will cause compilation warnings and may result in unexpected crashes. However, if a pointer is assigned the address of memory on the heap inside a function, then it can be accessed afterwards.

```
#include <stdio.h>
#include <stdlib.h>

int* fun() {
  int *i=0;
  i=(int*)malloc(sizeof(int));
  return i;
}

int main() {
  int *p = 0;
  p = fun();
  printf("%p\n",p);
  free(p);
  return 0;
}
```

After the function call, it is still necessary to call `free` to release the dynamically allocated memory.

Challenge problem

Write a program that uses dynamic allocation to request memory and then release it. Use the `sleep` function call to sleep for one second before calling `malloc` or `free`. Put the `malloc` and `free` function calls inside a loop, to generate a varying memory usage. Then use the examples given at the end of the C Cave in Issue 6 and the beginning of the C Cave in Issue 9, to monitor the memory usage of the test program.

GOING BALLISTIC

Target practice arcade game

W. H. Bell

MagPi Writer

Physics of a cannon ball

SKILL LEVEL : BEGINNER

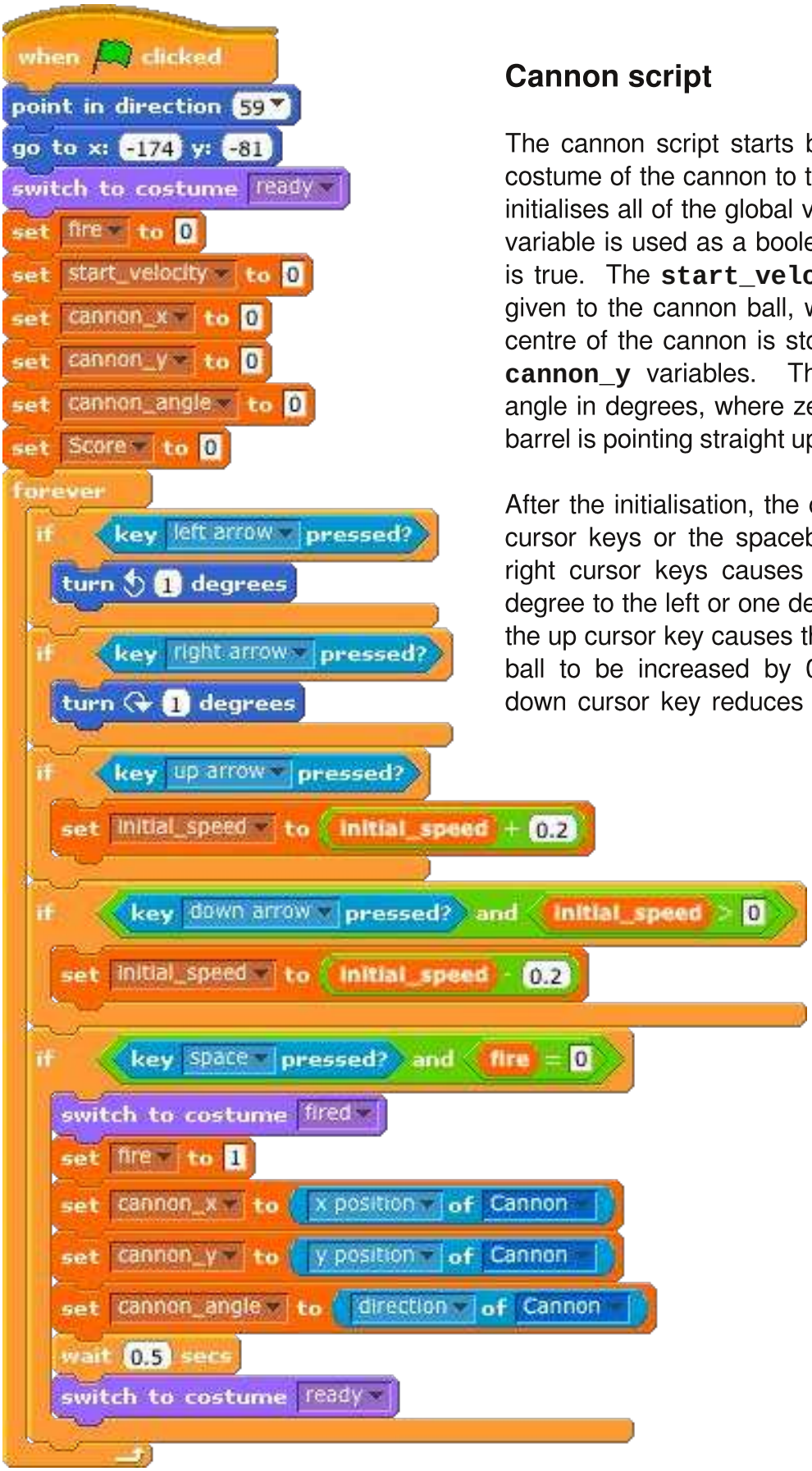
Welcome back to the Scratch Patch. This month's article is another simple arcade game. Similar to the game in Issue 13, there is plenty of scope for expanding or modifying this game too.

The idea of the game is to try to hit the drum with the cannon ball. The cannon is controlled by the cursor keys and the spacebar, where the left and right keys change the angle of the cannon, the up and down control the initial speed of the ball and the spacebar fires the cannon.

There are three sprites in this program: the cannon, the cannon ball and the drum. The cannon was drawn using the rectangle tool. Different shades of grey were used to give the appearance of a round barrel. Then the picture was copied using the Copy button. The original picture was named **`ready`** and the copy was named **`fired`**. A muzzle flash was added to the fired costume.

The cannon ball is a marble taken from the Scratch library. The Drum is also from the Scratch library. Both of these sprites were resized to the correct size.

The gun carriage is part of the stage, where the cannon was positioned on top within the cannon script.


when clicked
point in direction 59
go to x: -174 y: -81
switch to costume ready
set fire to 0
set start_velocity to 0
set cannon_x to 0
set cannon_y to 0
set cannon_angle to 0
set Score to 0
forever
if key left arrow pressed?
turn 1 degrees
if key right arrow pressed?
turn 1 degrees
if key up arrow pressed?
set initial_speed to initial_speed + 0.2
if key down arrow pressed? and initial_speed > 0
set initial_speed to initial_speed - 0.2
if key space pressed? and fire = 0
switch to costume fired
set fire to 1
set cannon_x to x position of Cannon
set cannon_y to y position of Cannon
set cannon_angle to direction of Cannon
wait 0.5 secs
switch to costume ready

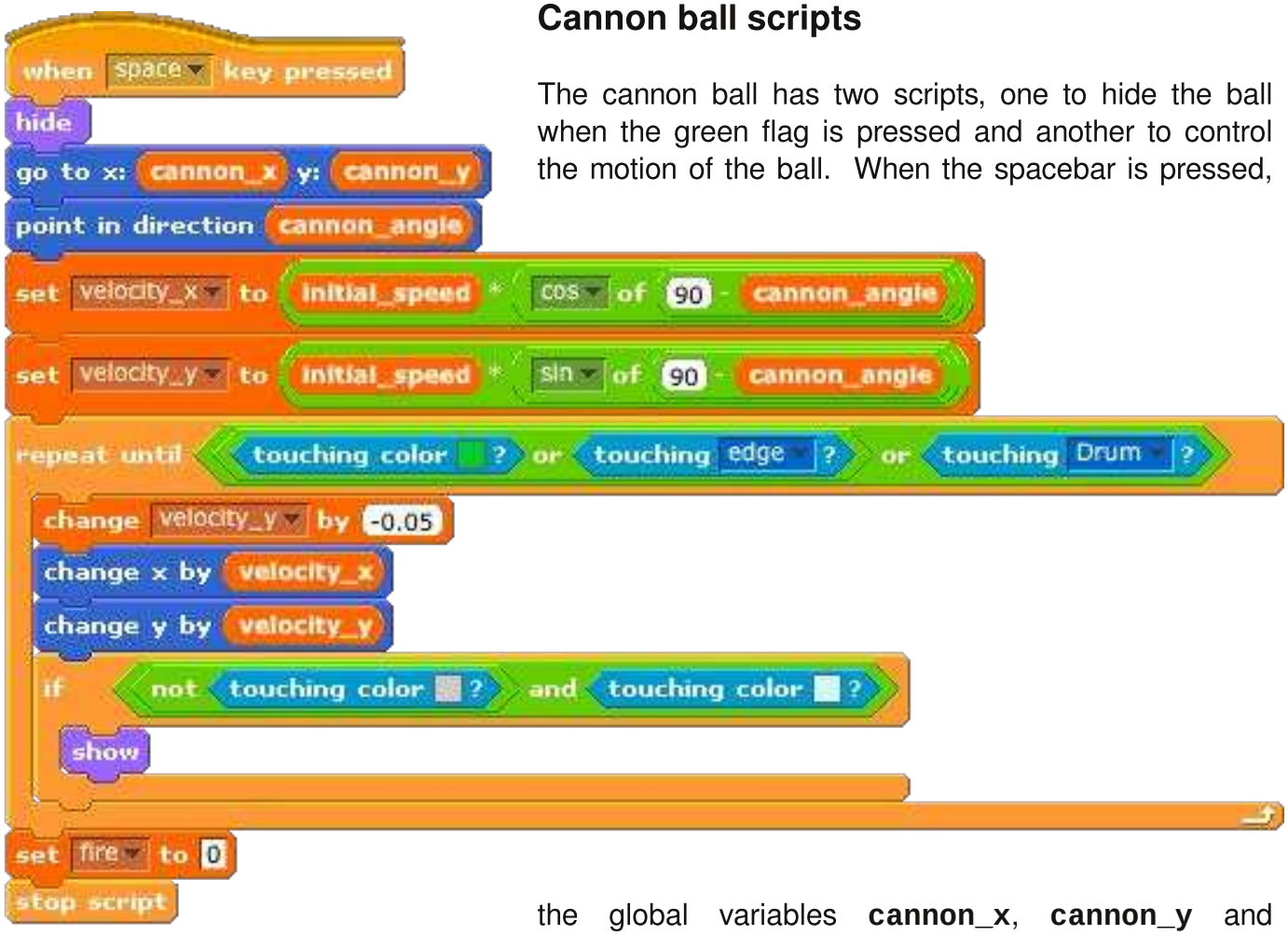
Cannon script

The cannon script starts by setting the position and costume of the cannon to the default. Then the script initialises all of the global variables with 0. The **`fire`** variable is used as a boolean, where 0 is false and 1 is true. The **`start_velocity`** is the initial velocity given to the cannon ball, when the ball is fired. The centre of the cannon is stored in the **`cannon_x`** and **`cannon_y`** variables. The **`cannon_angle`** is the angle in degrees, where zero implies that the cannon barrel is pointing straight up.

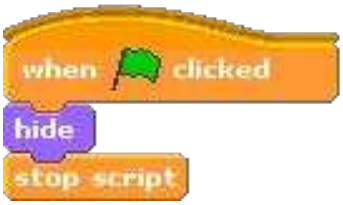
After the initialisation, the cannon reacts to either the cursor keys or the spacebar. Pressing the left and right cursor keys causes the cannon to rotate one degree to the left or one degree to the right. Pressing the up cursor key causes the initial speed given to the ball to be increased by 0.2, whereas pressing the down cursor key reduces the speed as long as it is greater than zero.

Pressing the spacebar will cause the cannon to fire if fire is also set to zero. The **`fire`** variable is used to prevent the cannon ball from being fired when the cannon ball is in flight. When the cannon is fired, the costume of the cannon is changed to show a flash. Then the **`fire`** variable is set to 1 and the global variables that describe the position of the cannon are set. After 0.5 seconds the flash is removed by resetting the costume.

Cannon ball scripts

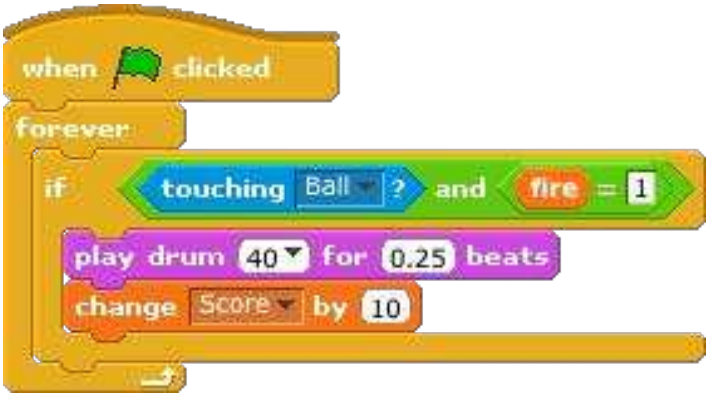

when space key pressed
hide
go to x: cannon_x y: cannon_y
point in direction cannon_angle
set velocity_x to initial_speed * cos of 90 - cannon_angle
set velocity_y to initial_speed * sin of 90 - cannon_angle
repeat until touching color ? or touching edge ? or touching Drum ?
change velocity_y by -0.05
change x by velocity_x
change y by velocity_y
if not touching color ? and touching color ?
show
set fire to 0
stop script

The cannon ball has two scripts, one to hide the ball when the green flag is pressed and another to control the motion of the ball. When the spacebar is pressed, the global variables **cannon_x**, **cannon_y** and **cannon_angle** are used to set the position of the ball and its initial velocity. Unlike other programming languages that use radians, Scratch uses degrees for the functions sine and cosine. The 90 degree offset is used to convert the angle of cannon, such that zero degrees corresponds to the cannon pointing horizontally to the right. The sine and cosine functions are then used to set the initial velocity components.



The cannon ball moves through the air until it touches either the colour green, the edge of the stage or the Drum. While none of these conditions are met, the vertical velocity component is decremented by 0.05. This decrementation corresponds to the acceleration due to gravity. Next the current x and y components of the velocity are used to move the ball through the air. The speed with which the repeat-until loop executes corresponds to the time component.

The ball is hidden unless it has completely cleared the cannon barrel. This is achieved by checking if the ball is not touching grey and is touching the colour of the sky. Lastly, when the script finishes, the global variable fire is set to zero to allow the cannon to be fired again.



The drum script

There is one drum script, which is launched when the green flag is pressed. When the ball is launched, the global variable **`fire`** remains set to one until the ball touches down on the ground or the drum. Therefore, combining the **`fire`** variable requirement with touching the ball sprite implies that the program will enter the if statement once for one cannon ball flight. Once the program has entered the if condition, the drum plays a simple sound and increments the global variable **`score`** by 10 points.

Rotating the cannon

In order for the cannon to be rotated around the centre of the cannon sprite, the **`Set costume center`** button should be clicked. Then put the crosshairs for both costumes at the middle of the sprite.

Projectile motion summary

The motion of a projectile can be split into vertical and horizontal components. Gravity acts in the vertical direction as a downward acceleration. In the program documented in this article, air resistance is neglected. Therefore, the horizontal velocity component is constant and the horizontal distance the projectile travels is found by multiplying the initial horizontal velocity by the time it is in the air. The time the projectile is in the air is determined as the difference between the start time and when the vertical position is equal to the ground. While the projectile is in flight, gravity continues to decrement the vertical velocity until the projectile hits the ground. If the initial vertical velocity component is small, it will not take long for gravity to bring the projectile down. Notice that within this simple program the vertical position is evaluated in two steps, rather than in one equation. There is one step to evaluate the change in velocity and another to update the horizontal position.

Possible extensions

The drum could be replaced with other objects that move around to make the game harder. There could be a time limit for the player to hit all of the targets with the cannon. The equation of motion used in this game does not include drag. Drag can be expressed as a constant force for a given velocity squared. The problem of estimating the drag could be made harder by introducing wind, which would cause the relative velocity of the ball with respect to the air to be higher.

Feedback & Question Time

We bought a Raspberry Pi a few months ago, but didn't really know what to do with it. We wanted something to learn as a project between my son and I. Initially we (predictably) installed media centres, but couldn't get my son interested. He was one of the Facebook zombies. I happened to drop on your mag from the Raspberry Pi website and thanks to you I now have a son that has written his first Python script and is very keen to learn more. We are now collecting parts to create our own robot with camera. Facebook's loss is Raspberry Pi's gain!

Thank you, thank you.

Scott

P.S. We are learning together so I have also written my first Python script!

My first article in @TheMagP1 as guest writer (page 12, issue 16) YAY! Pi #MoreThanHappy

Jorge Rance

I got my issues and binder a while ago (from the Kickstarter campaign) and they are so cool...thank you.

Randy

I have found The MagPi very helpful, as I am a Raspberry Pi beginner myself. You helped me set up CUPS printing. :)

When friends tell me they are getting a Raspberry Pi, the two things I recommend for them are Berryboot and the MagPi!

Thanks,
Doug Smith

Dear Editor,

I love playing with my Raspberry Pi (setting up webcams, learning Python, etc.) and really enjoy doing the projects featured in The MagPi. I am interested in learning electronics and would like to incorporate this into future projects with my Raspberry Pi, but I don't know where to start. Could you, or anyone, give me any advice on where to begin please?

Many thanks,
Claire

[Ed.: Can you help Claire out? If so please contact us with your suggestions and ideas. Thank you.]

What do you think of The MagPi and the Raspberry Pi? Have you got a project idea and do not know where to start? Why not write to The MagPi and tell us what is on your mind?

The MagPi is a trademark of The MagPi Ltd. Raspberry Pi is a trademark of the Raspberry Pi Foundation. The MagPi magazine is collaboratively produced by an independent group of Raspberry Pi owners, and is not affiliated in any way with the Raspberry Pi Foundation. It is prohibited to commercially produce this magazine without authorization from The MagPi Ltd. Printing for non commercial purposes is agreeable under the Creative Commons license below. The MagPi does not accept ownership or responsibility for the content or opinions expressed in any of the articles included in this issue. All articles are checked and tested before the release deadline is met but some faults may remain. The reader is responsible for all consequences, both to software and hardware, following the implementation of any of the advice or code printed. The MagPi does not claim to own any copyright licenses and all content of the articles are submitted with the responsibility lying with that of the article writer. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit:

http://creativecommons.org/licenses/by-nc-sa/3.0/

Alternatively, send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.